Getting Started with Asio

An Asio Based Flash XML Server

Michael Caisse

Object Modeling Designs
www.objectmodelingdesigns.com
Copyright © 2010

BoostCon, 2010

1. Introducing ASIO
   - Asynchronous I/O
   - Asio Basics
2. Communication with ASIO
   - Buffers
   - API
3. Flash XML Server
   - The Goal
   - The Server Class

Outline

1. Introducing ASIO
   - Asynchronous I/O
   - Asio Basics
2. Communication with ASIO
   - Buffers
   - API
3. Flash XML Server
   - The Goal
   - The Server Class

What is Asio

An Asynchronous I/O Library

- Started as a network library. Now supports a variety of resources:
  - Serial Ports
  - Timers
  - File Descriptors
- Uses an efficient Proactor model
- Exremely Scalable - Easily supporting thousands of connections.
- Provides a Portable Abstraction

What is Asynchronous I/O

Daughter #1

me:	“Please make me a coffee.”
daughter:	“Sure Dad”

time passes ... I work. She makes a cappuccino.

daughter:	“Here is your coffee.”
me:	“Thanks”

What is Asynchronous I/O

Daughter #3

me:	“Please make me a coffee.”
daughter:	“I would love to!”

we both walk to the machine. I supervise (watch). She makes a cappuccino.

daughter:	“Here is your coffee.”
me:	“Thanks”

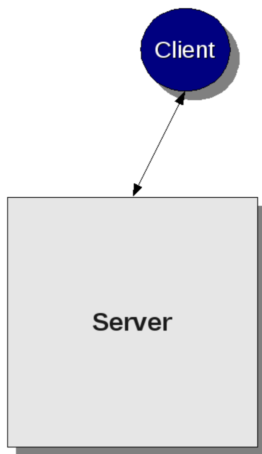
What is Asynchronous I/O

```
void done_reading()
{...}

read_file( filename, buffer, done_reading );

... do work
```

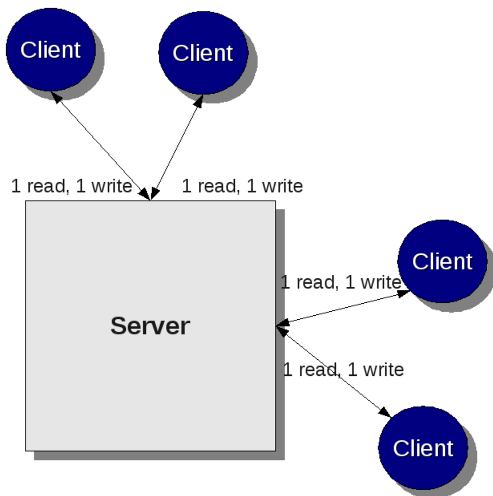
Why Asynchronous I/O?

Sounds hard ... why do it?

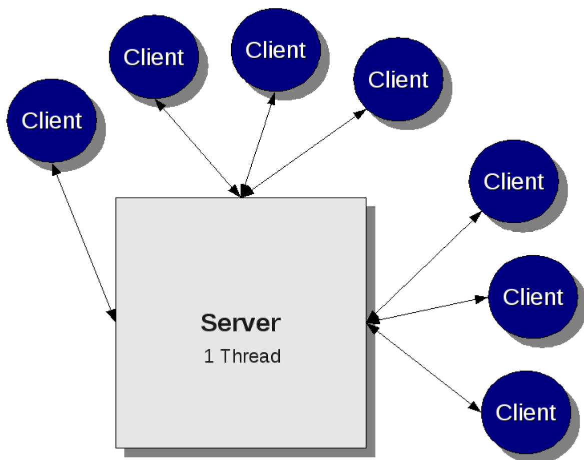
Why Asynchronous I/O?



Why Asynchronous I/O?



Asio Asynchronous



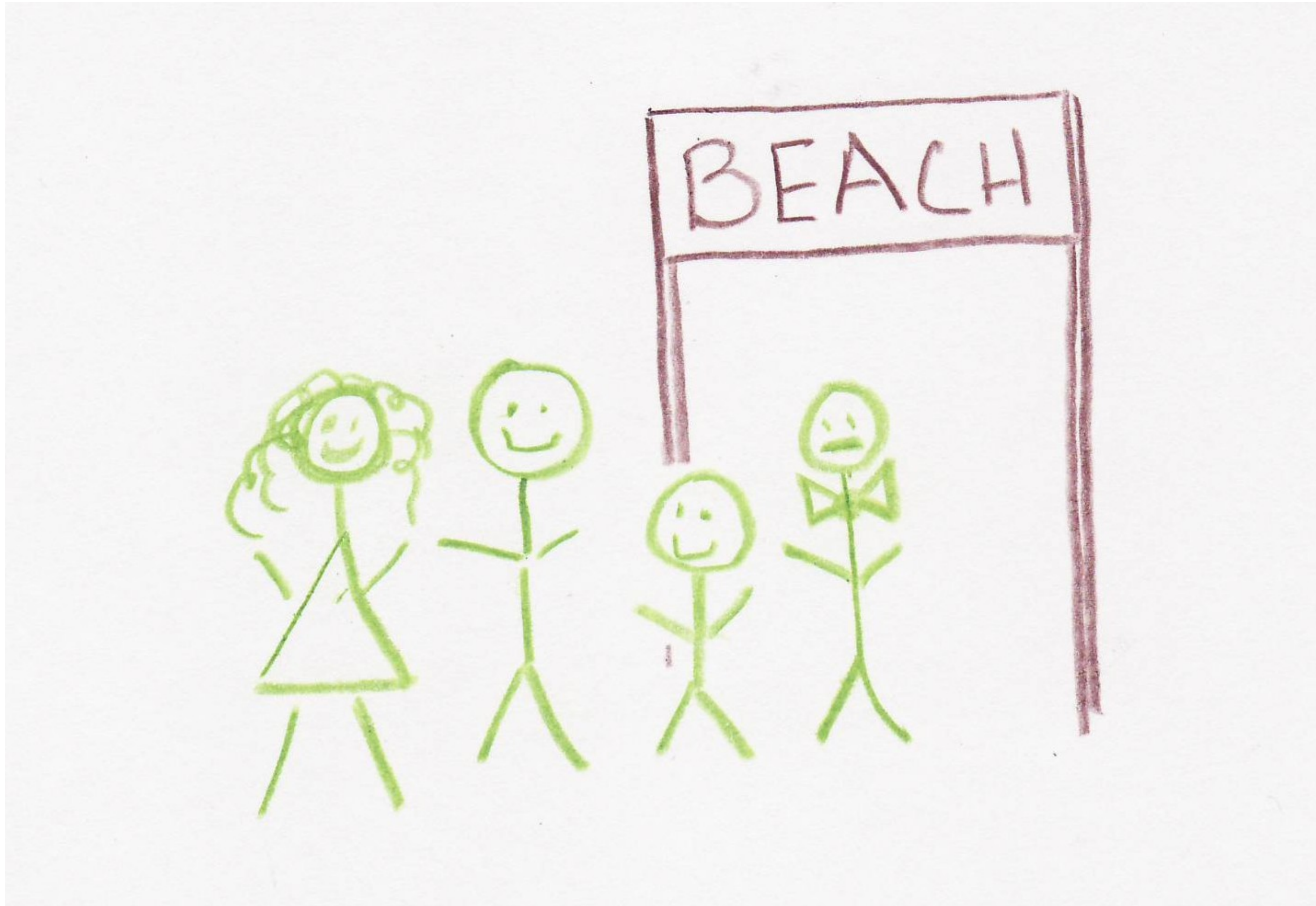
A Proactor Story

A Proactor Story

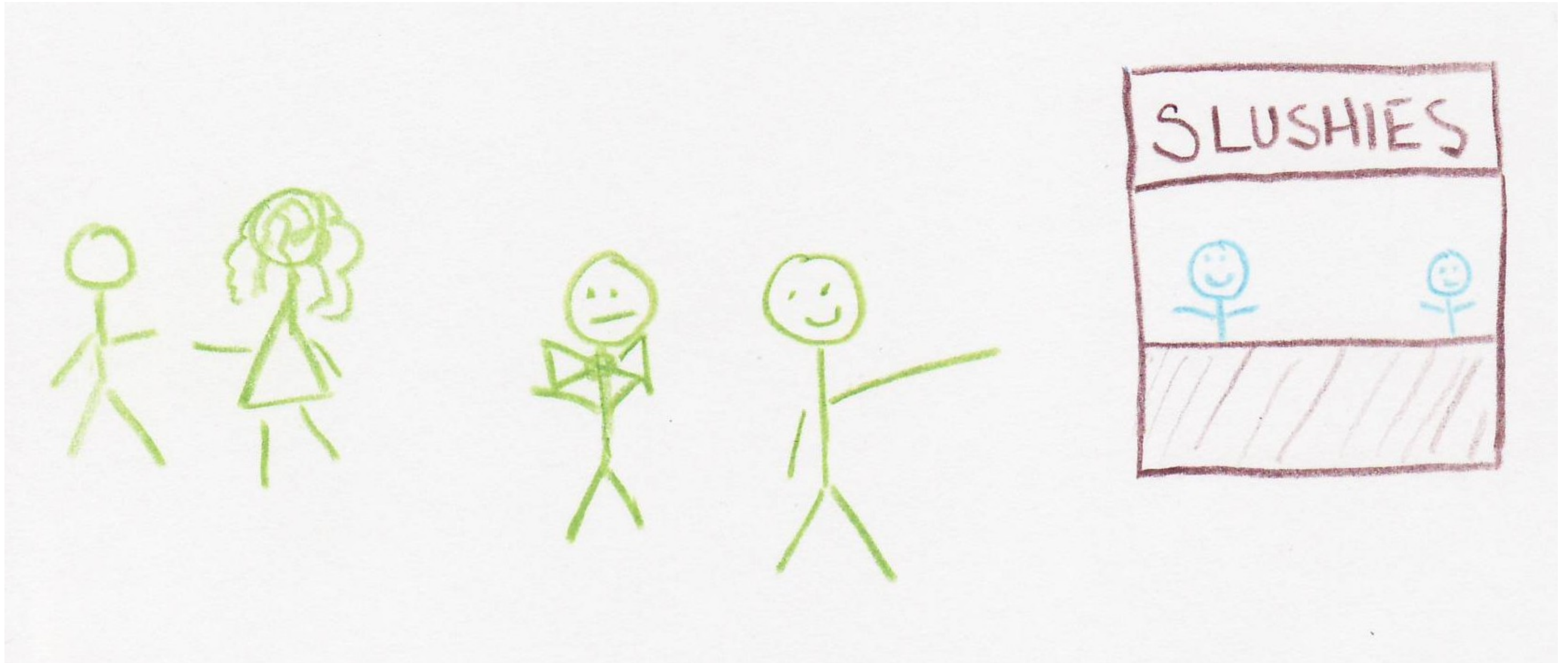
- or -

Purple Slushies, Butlers and
Brain Freeze

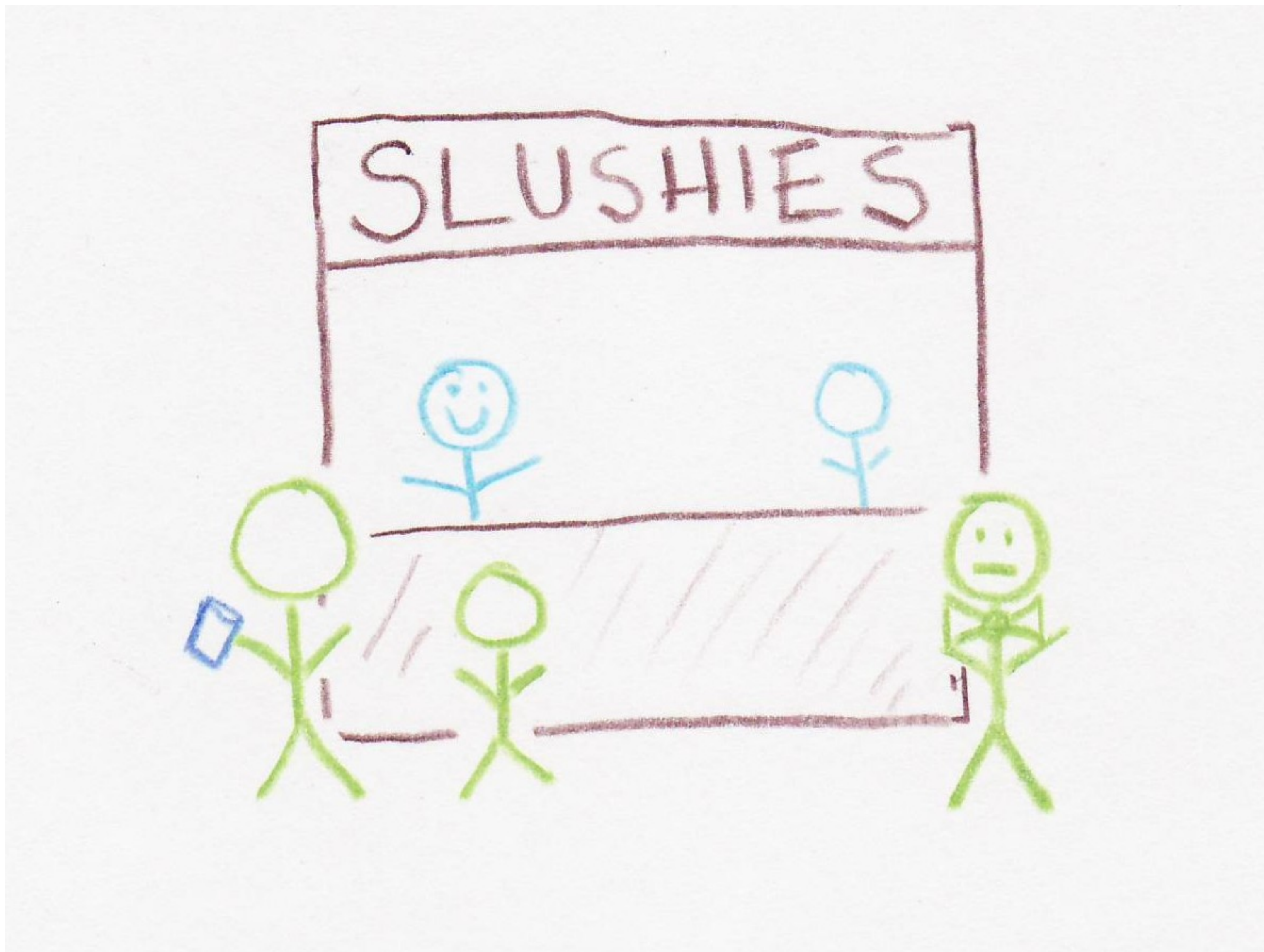
Copyright 2010 Object Modeling Designs



Mom, Dad, Johnny and Butler go to the beach.

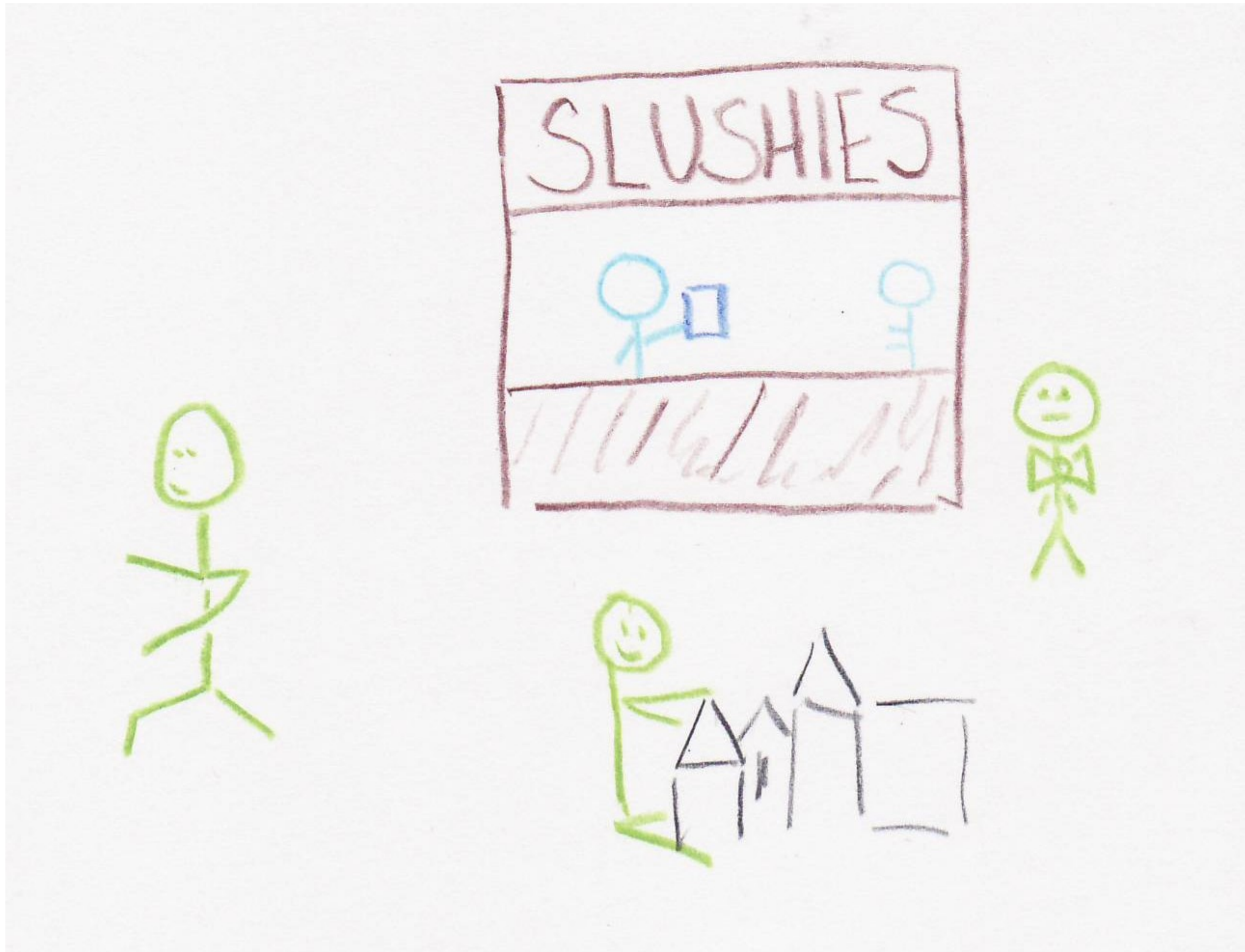


Dad tells Butler to wait at the Slushie Shack.



After some time, Dad and Johnny go to get a slushie. Dad brings his own cup. He is greeted by the Owner.

“I would like to order a slushie. Here is my cup.
Please deliver it to Johnny when it is ready.

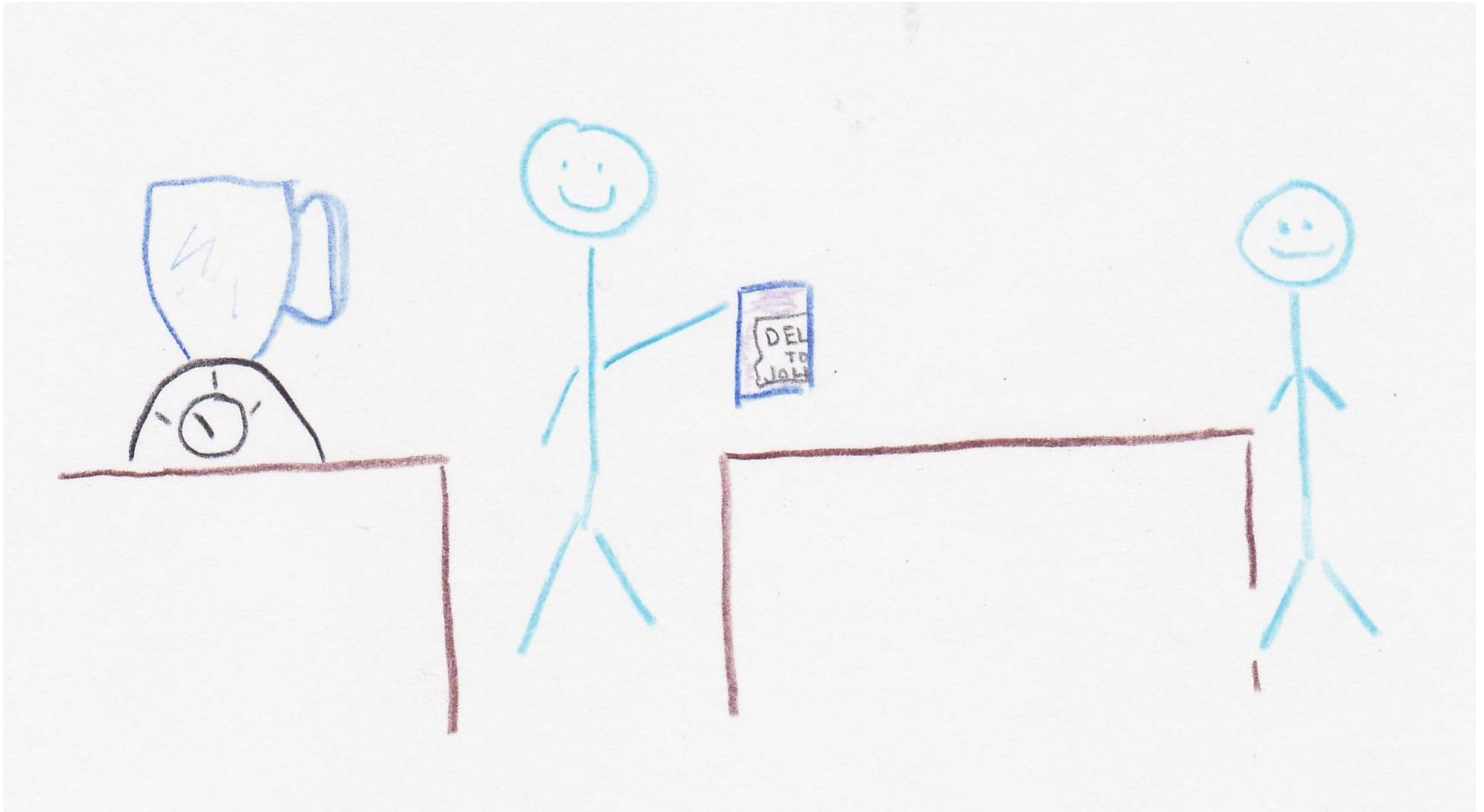


Dad heads off to explore the beach. Johnny builds a sandcastle. Owner begins to make the slushie.
And Butler waits.

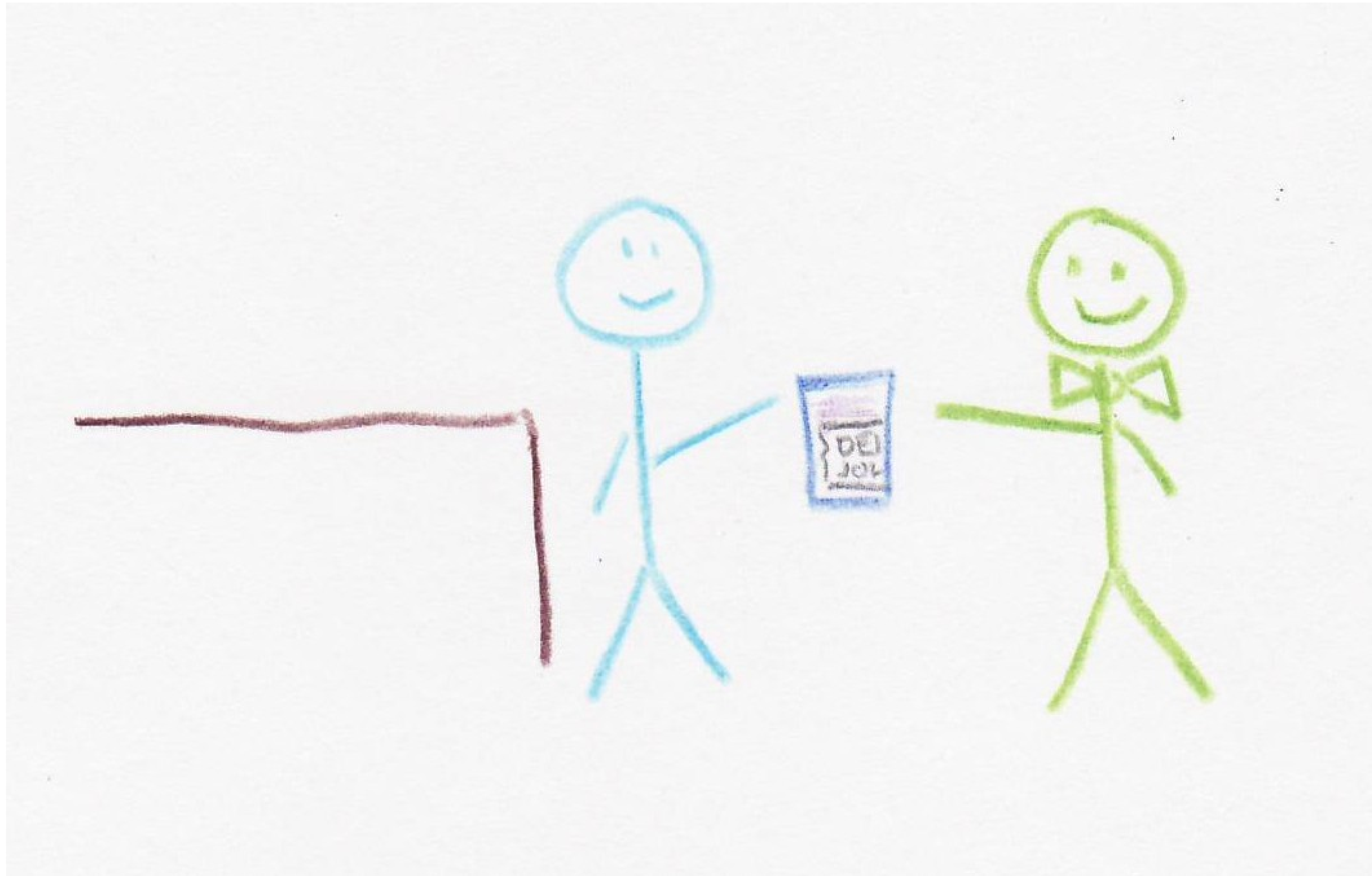
Owner starts the blender and goes back to take the next customer's order.

… *<time passes>* …

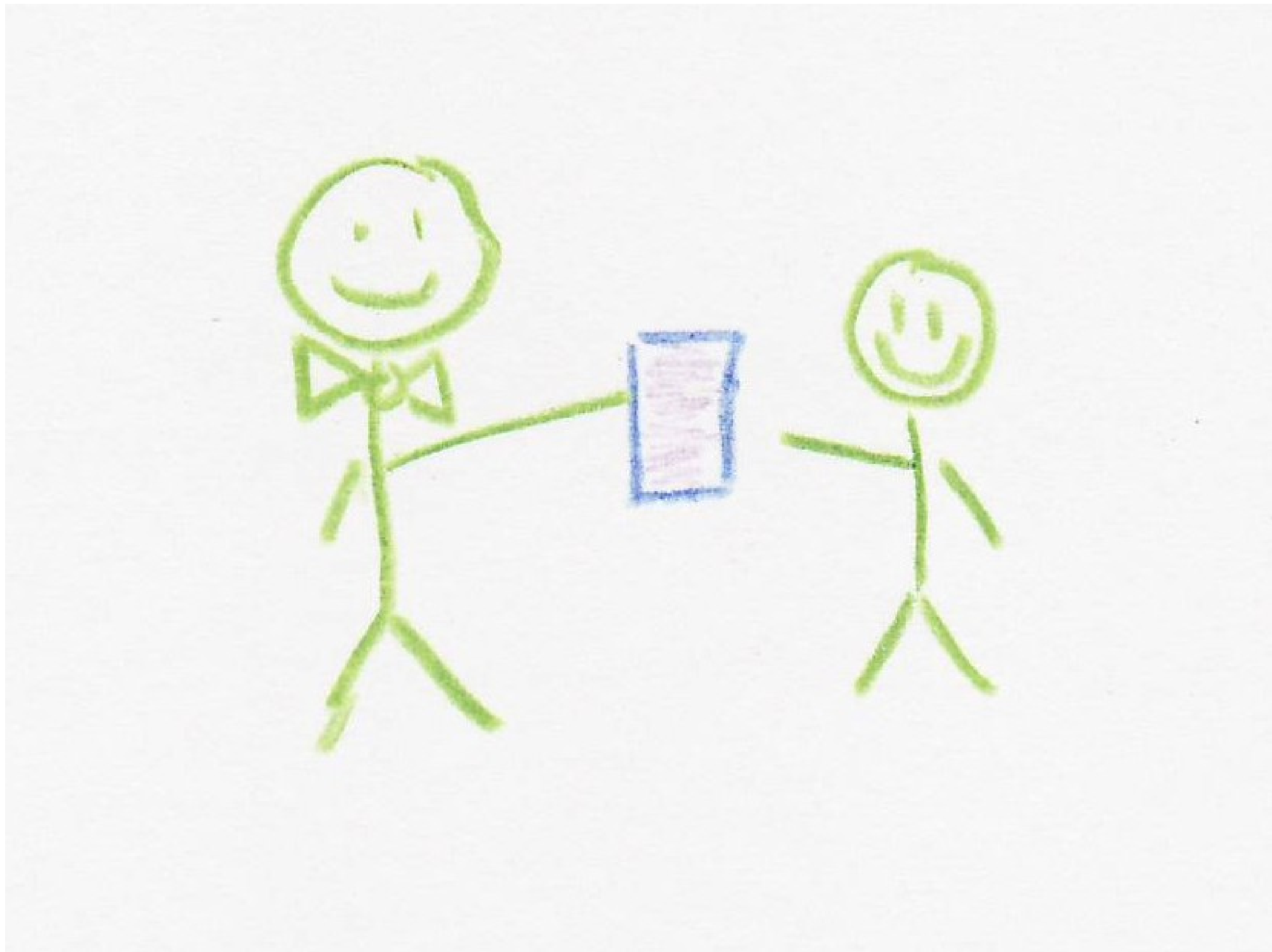
“Ding”



Slushie is ready and Owner moves the cup to the completion table where Assistant is waiting.



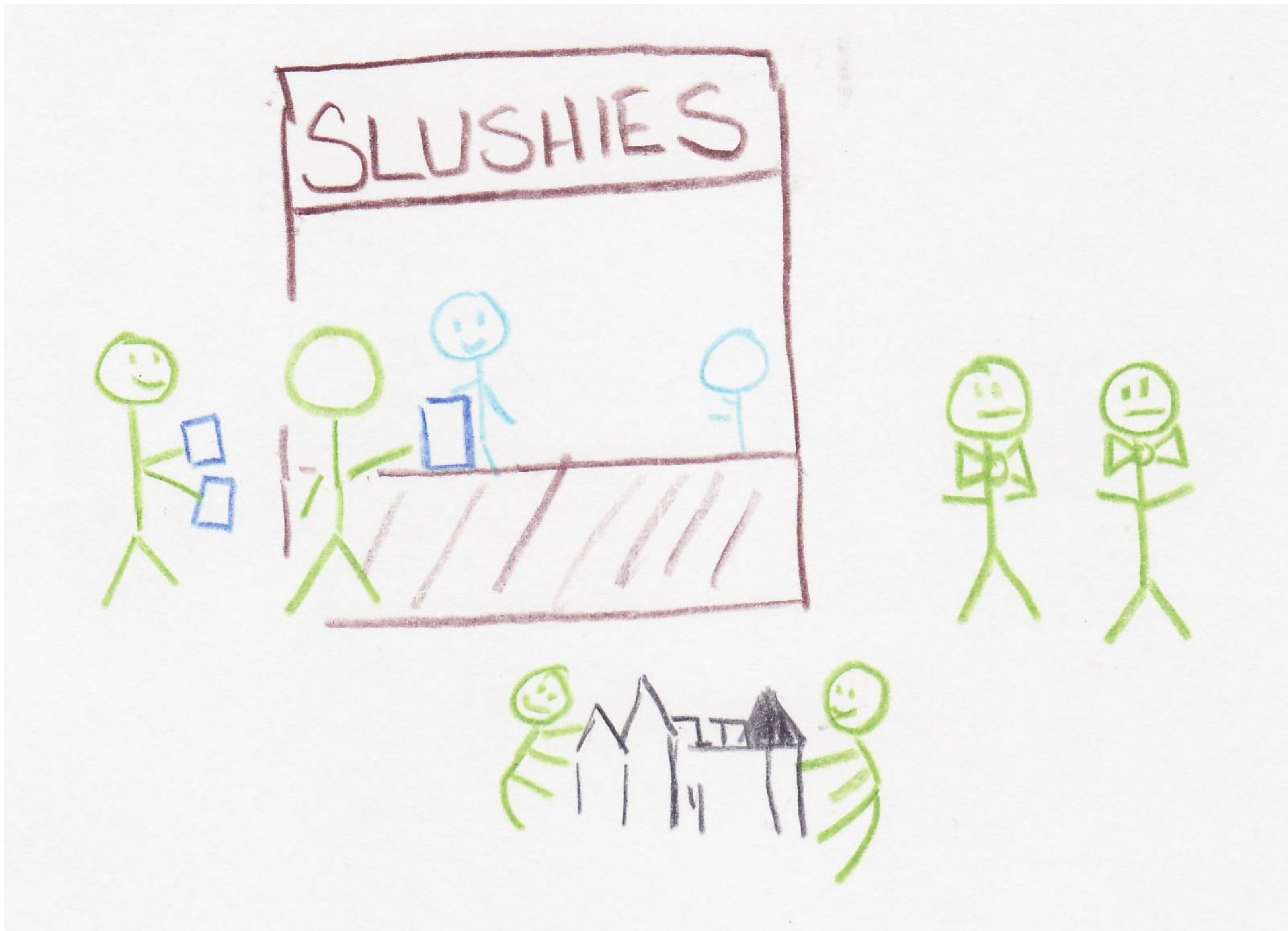
The Assistant gives the slushie to Butler for delivery to Johnny. Butler is happy to have something to do.



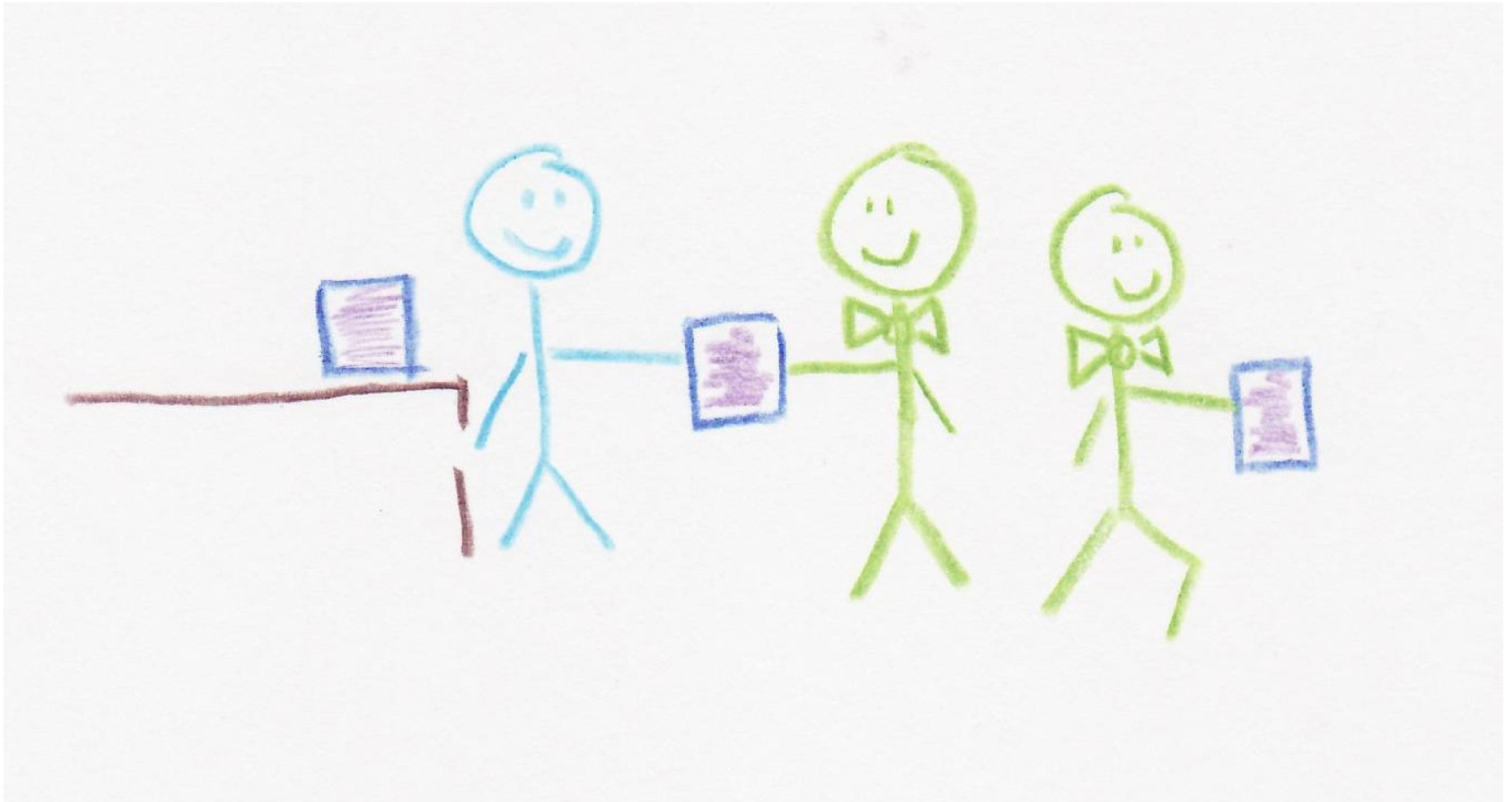
Butler delivers the slushie to Johnny who is happy too. Butler returns to the Slushie Shack and waits.

Sometimes Dad will order multiple slushies.
One for Mom and one for Johnny.

That isn't a problem. Assistant just gives the first one ready to Butler. Bulter can only deliver one at a time and returns for the second slushie.

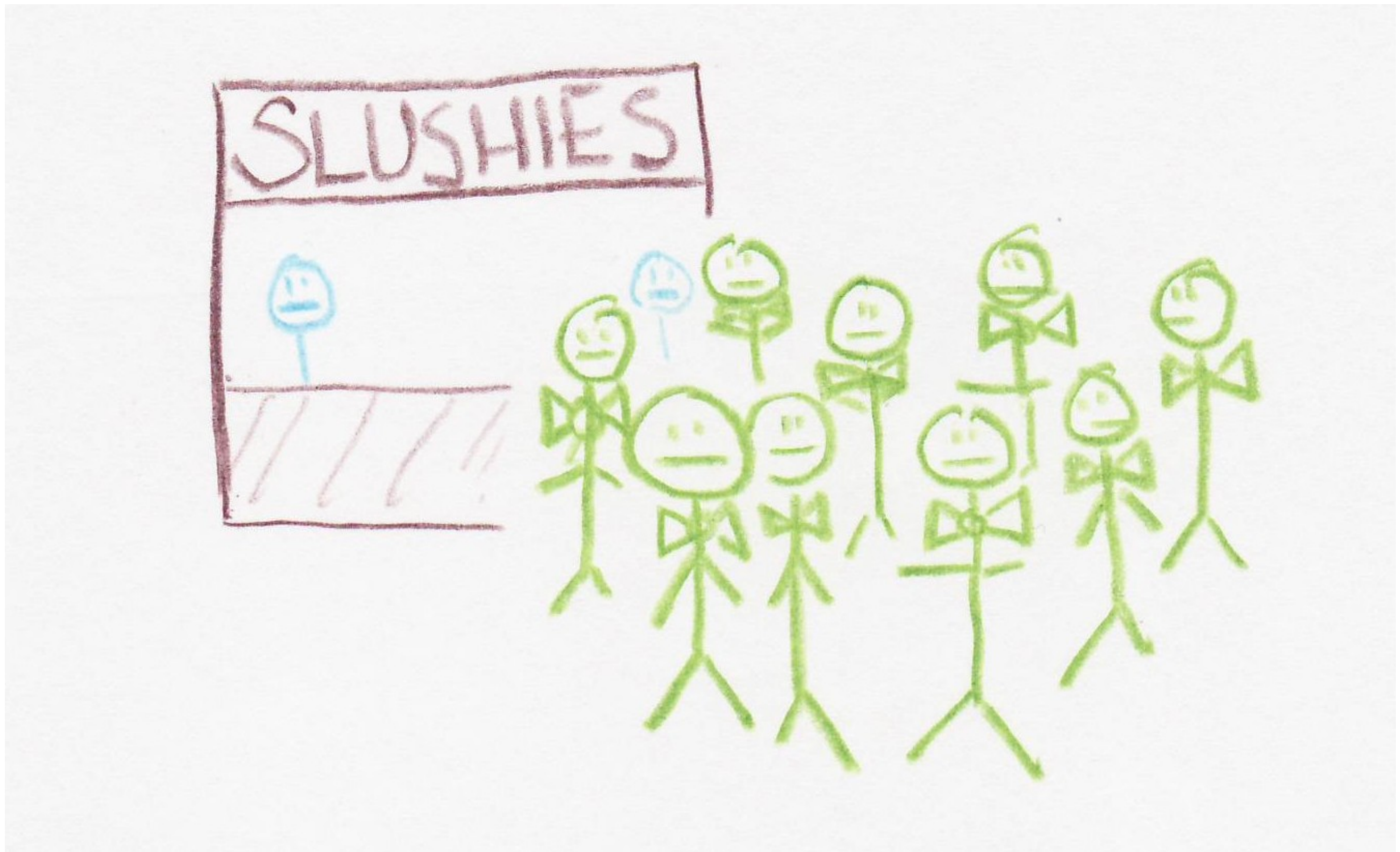


Other families come to the beach and bring their butlers who also wait in the slushie completion line.

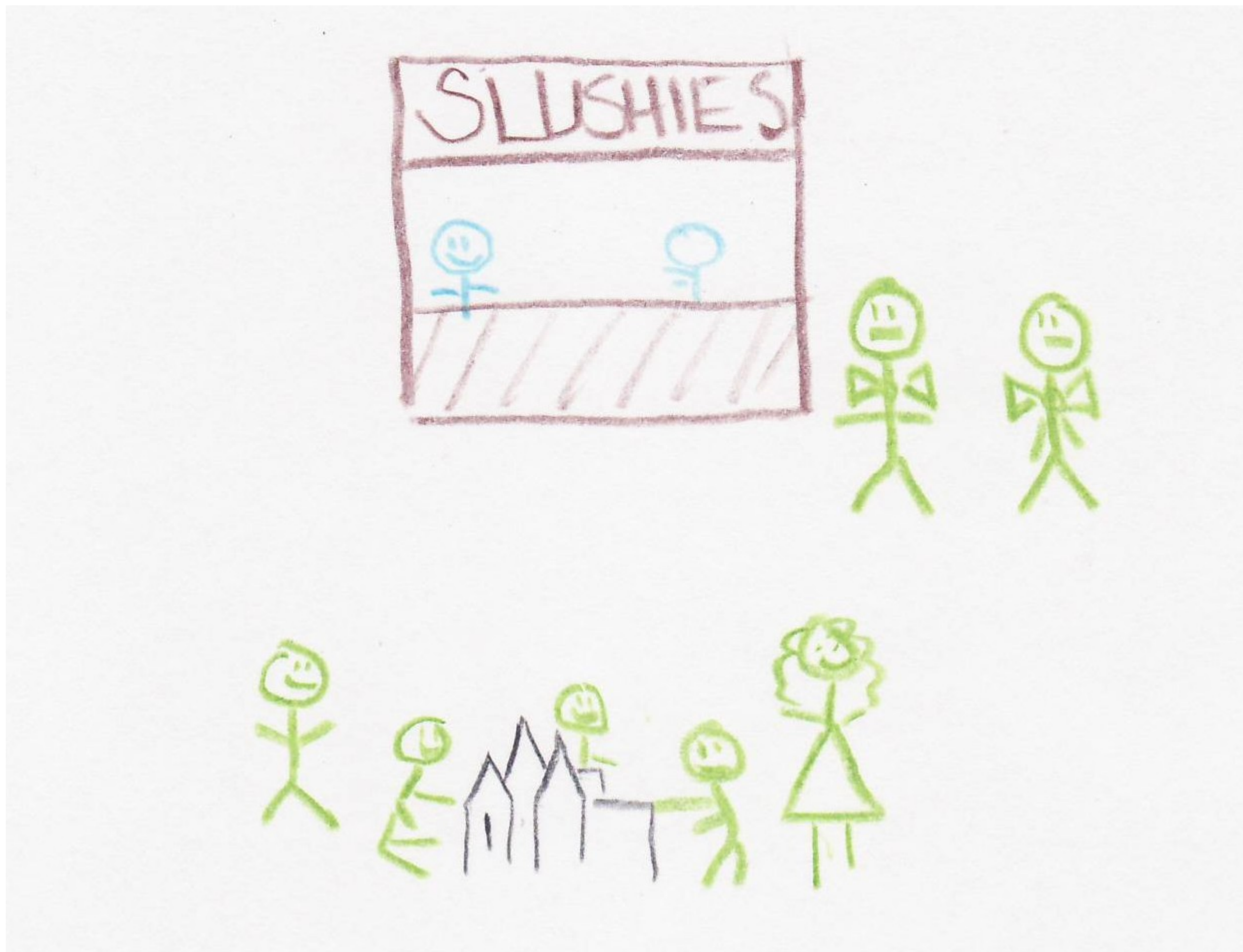


This works well because it helps keep Assistant's slushie completion table empty.

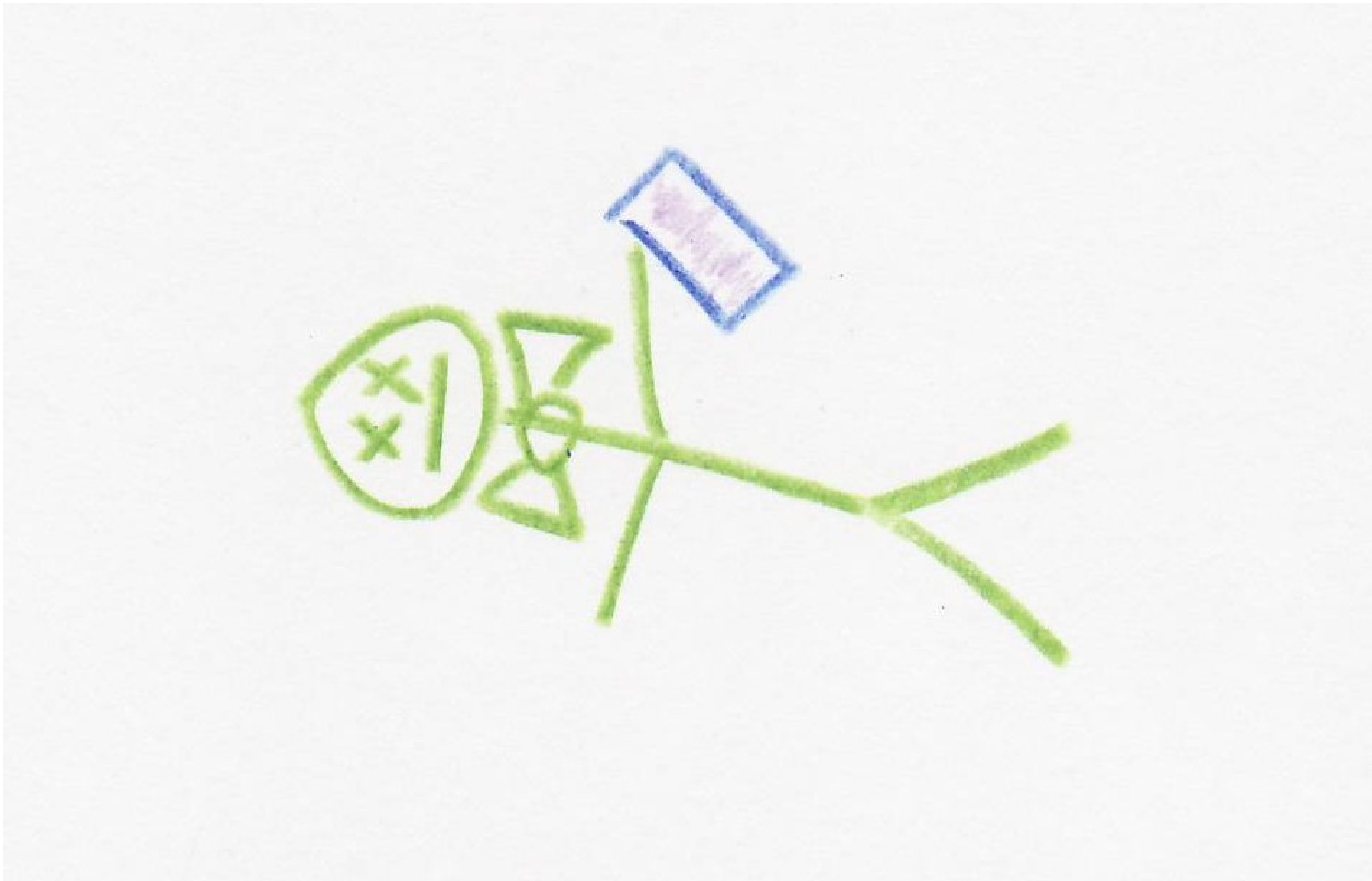
Assistant still remembers that fateful day when no butlers came to the beach.



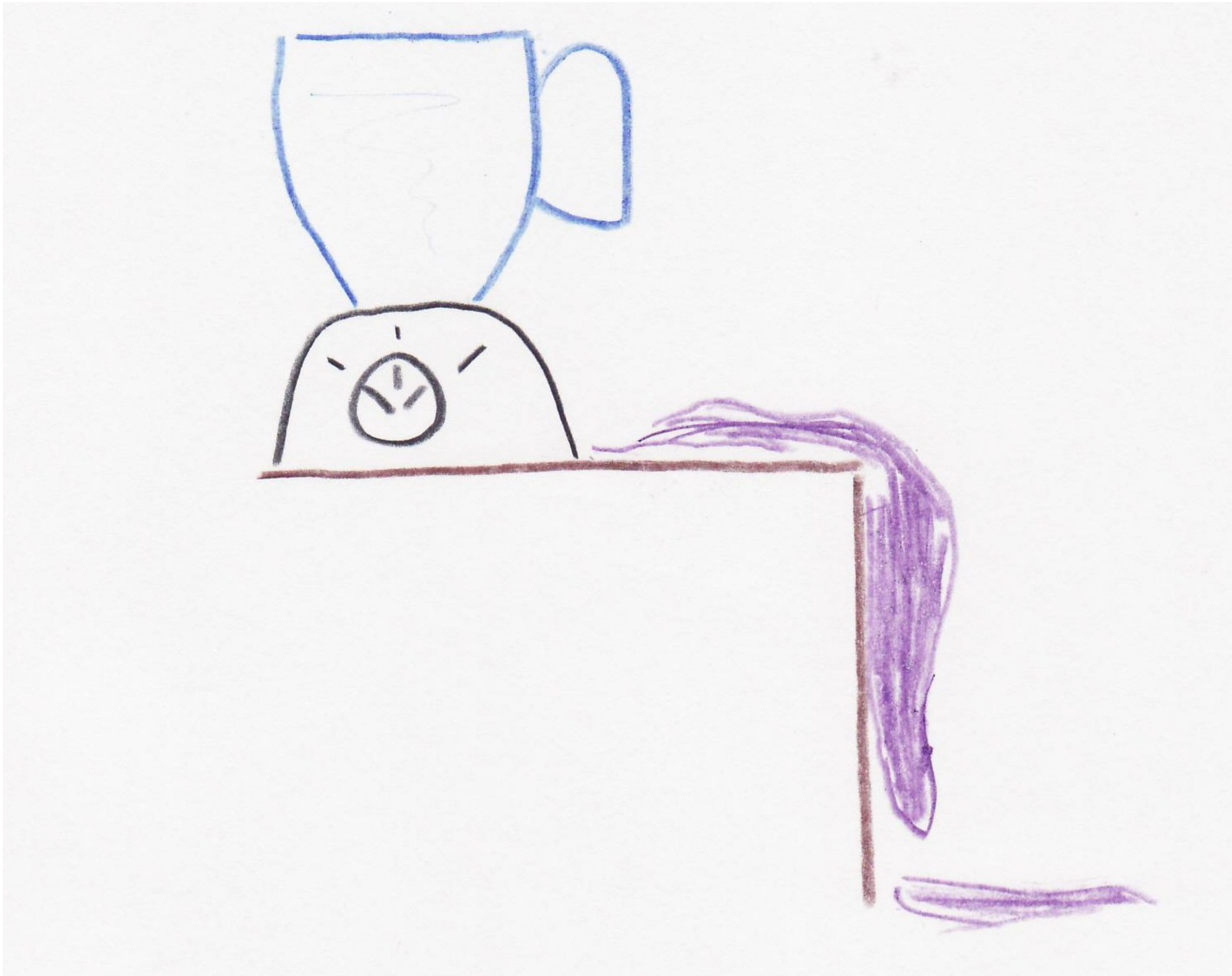
There was also the time that each kid brought a butler. Disaster! No room at the shack. Too busy, yet nothing was getting done.



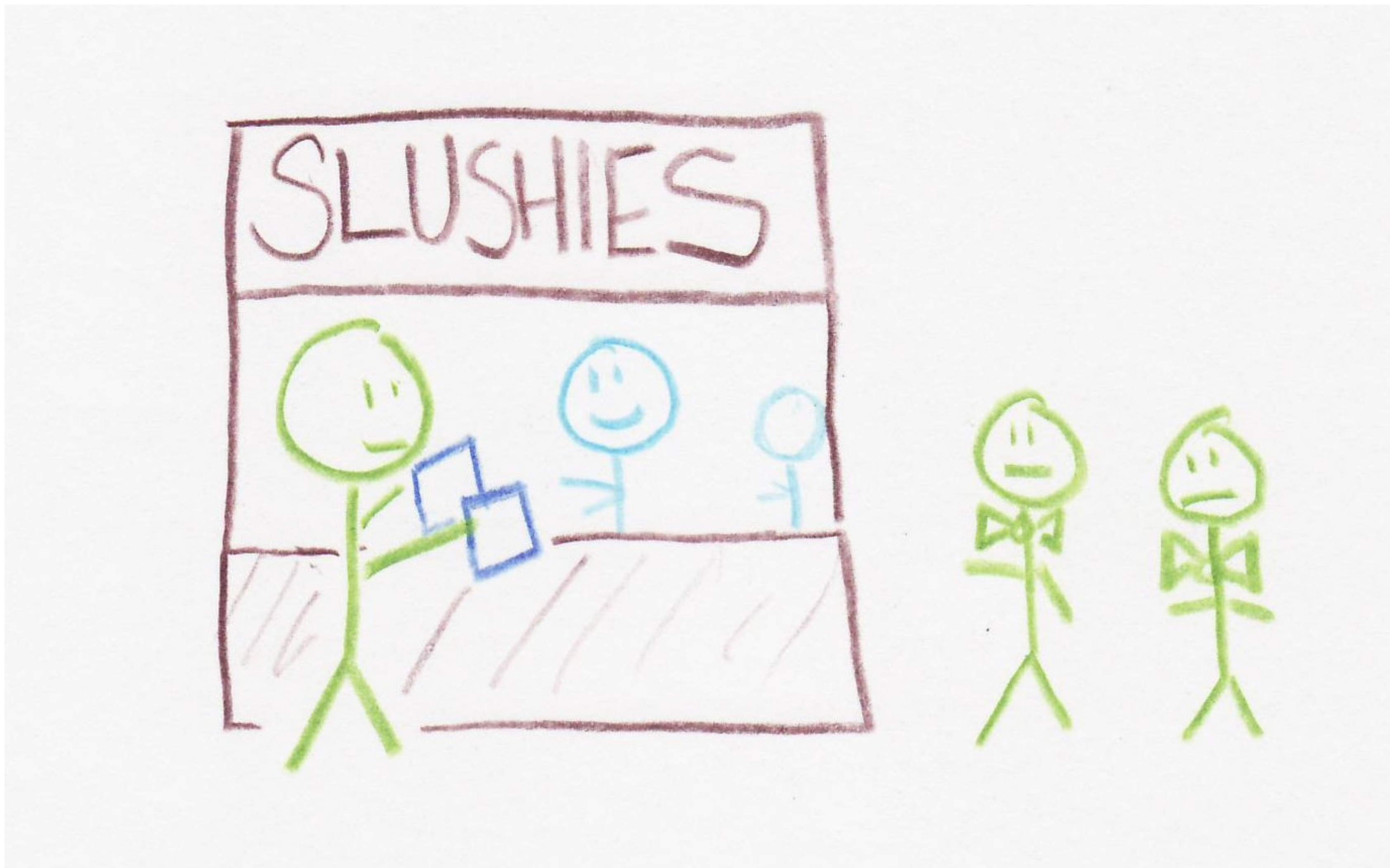
The families agreed that two butlers would be plenty for all. They now share.



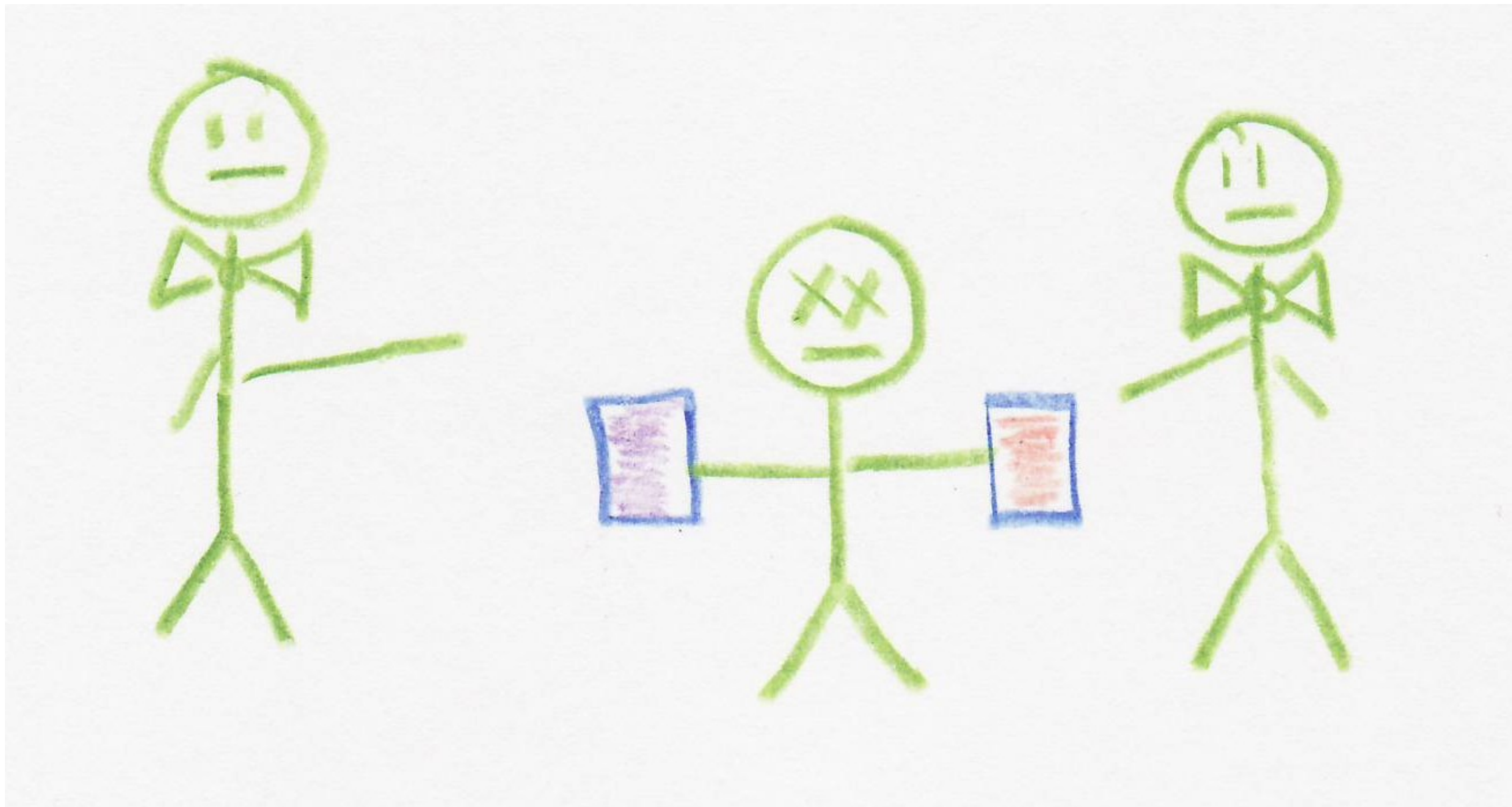
Occasionally tragedy strikes. Johnny will leave to chase waves without getting his slushie. Butler will die of exhaustion trying to find him.



… or somebody will take their cup and go home while the slushie is being made. Then it gets poured on the floor. Yuck.

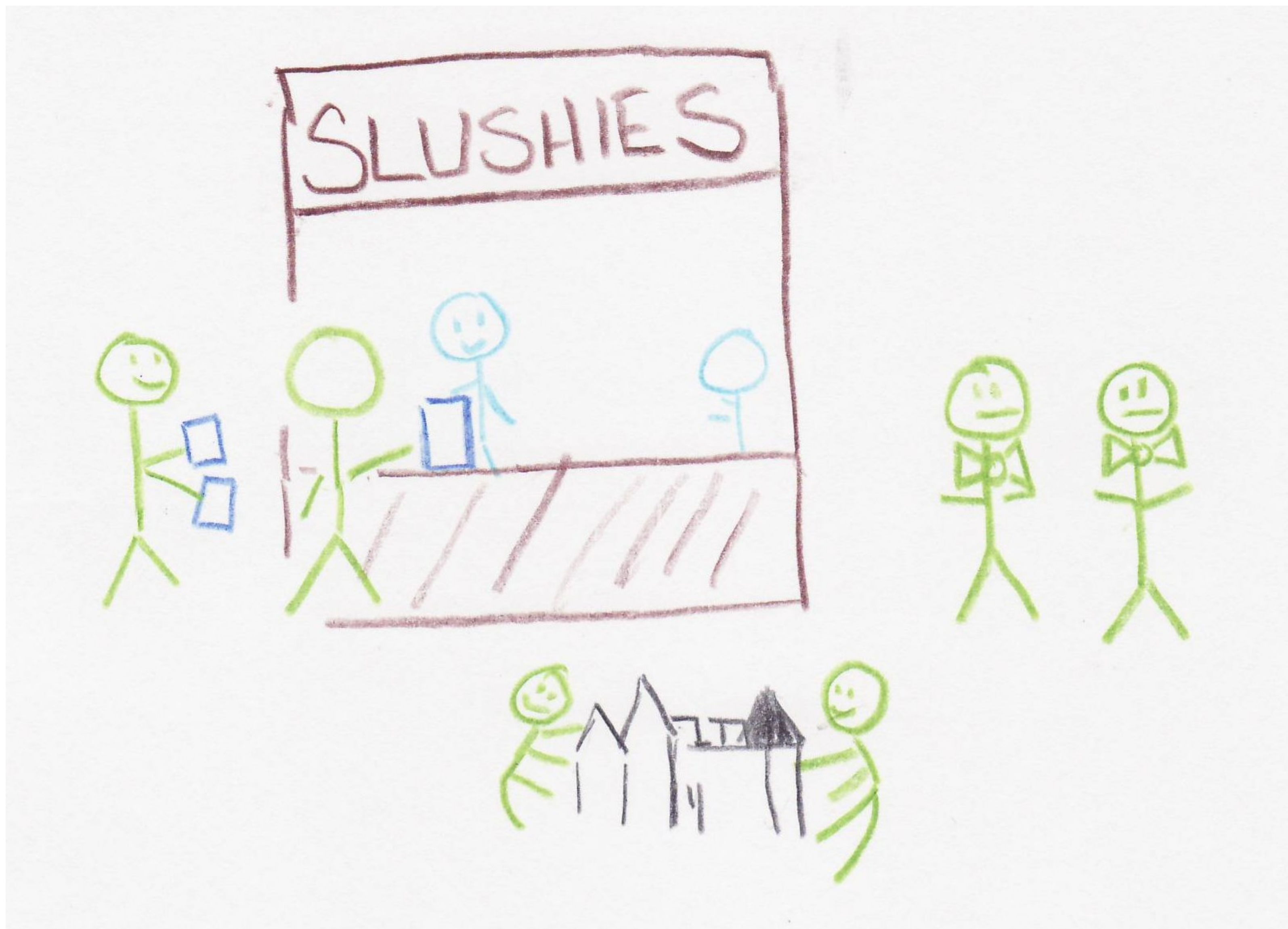


Dad is sometimes very generous. “Johnny would like one orange and one purple slushie.”



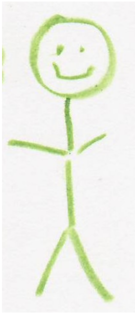
If both slushies are done at the same time and both butlers are available then Johnny gets two slushies at once. This confuses Johnny and causes brain freeze.

Susie is smarter and doesn't mind both slushies at one time.



But most often the dads are making requests to the Owner, the Assistant is monitoring the table, the kids are building sandcastles and
the butlers are waiting.

Credits



Initiator
(Dad)

Asynchronous
Operation Processor
(Owner)

Proactor
(Butler)

Asynchronous
Operation
(Blender Making
Slushies)

Asynchronous
Event Demultiplexer
(Assistant)

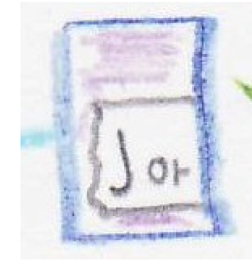
Completion
Event Queue
(Completion Table)

Completion Handler
(Johnny)

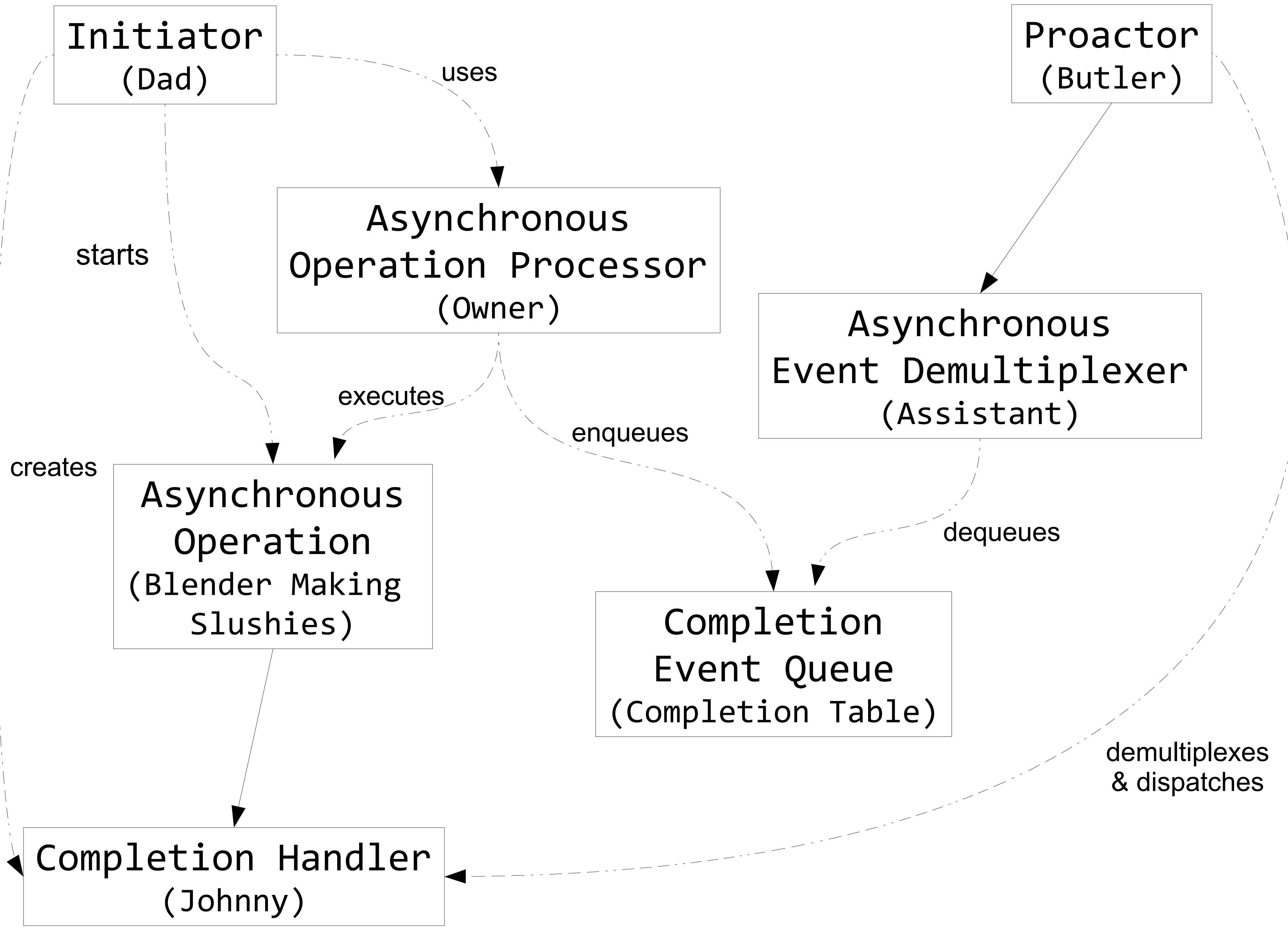
Additional Roles

Operating System
(Blender)

Memory to
be Filled
(Empty Cup)



Data in Memory
(Full Cup)



Initiator
(Dad)
Proactor
(Butler)
uses
starts
Asynchronous
Operation Processor
(Owner)
Asynchronous
Event Demultiplexer
(Assistant)
executes
enqueues
creates
Asynchronous
Operation
(Blender Making
Slushies)
dequeues
Completion
Event Queue
(Completion Table)
demultiplexes
& dispatches
Completion Handler
(Johnny)

Some Lessons

- All threads of activity in the Slushie Shack stayed in the Slushie Shack
- The Butler delivered the results to the completion handler
- The Butler (handler thread) was supplied by the family (application)
- The cup (memory) was supplied and owned by the family (application)

More Lessons

- Not all handlers (Johnny) liked having multiple results delivered at the same time
- Some handlers (Susie) didn't care if they had multiple results delivered at once
- Don't leave the beach (scope) when a slushie is being made for you
- A few handler threads (butlers) can service many completion routines

Outline

1. Introducing ASIO
   - Asynchronous I/O
   - Asio Basics
2. Communication with ASIO
   - Buffers
   - API
3. Flash XML Server
   - The Goal
   - The Server Class

Simple Timer

```
 1  // simple_timer.cpp
 2  void timer_expired( const boost::system::error_code& e )
 3  {
 4     std::cout << "timer expired." << std::endl;
 5  }
 6
 7  int main()
 8  {
 9     boost::asio::io_service asio_service;
10
11     boost::asio::deadline_timer timer( asio_service,
12                                        boost::posix_time::seconds(5) );
13
14     timer.async_wait( timer_expired );
15
16     std::cout << "calling io_service::run" << std::endl;
17
18     asio_service.run();
19
20     std::cout << "done." << std::endl;
21
22     return 1;
23  }
```

Simple Timer

Output

```
calling io_service::run
timer expired.
done.
```

```
1  // simple_timer.cpp
2  void timer_expired( const boost::system::error_code& e )
3  {
4     std::cout << "timer expired." << std::endl;
5  }
6
7  int main()
8  {
9     boost::asio::io_service asio_service;
10
11    boost::asio::deadline_timer timer( asio_service,
12                                       boost::posix_time::seconds(5) );
13
14    timer.async_wait( timer_expired );
15
16    std::cout << "calling io_service::run" << std::endl;
17
18    asio_service.run();
19
20    std::cout << "done." << std::endl;
21
22    return 1;
```

Simple Timer with Timings

```
1  void timer_expired( const boost::system::error_code& e )
2  {
3      std::cout << now_time << " : timer expired." << std::endl;
4  }
5
6  int main()
7  {
8      asio::io_service asio_service;
9
10     asio::deadline_timer timer( asio_service,
11                                 boost::posix_time::seconds(5) );
12
13
14     std::cout << now_time << " : request async_wait 5-seconds" << std::endl;
15     timer.async_wait( timer_expired );
16
17     std::cout << now_time << " : sleep 3-seconds" << std::endl;
18     boost::this_thread::sleep( boost::posix_time::seconds(3) );
19
20     std::cout << now_time << " : calling io_service::run" << std::endl;
21     asio_service.run();
22
23     std::cout << now_time << " : done." << std::endl;
24
25     return 1;
26 }
```

Simple Timer with Timings

Output

```
2010-May-08 19:04:13 :  request async_wait 5-seconds
2010-May-08 19:04:13 :  sleep 3-seconds
2010-May-08 19:04:16 :  calling io_service::run
2010-May-08 19:04:18 :  timer expired.
2010-May-08 19:04:18 :  done.
```

```
1   void timer_expired( const boost::system::error_code& e )
2   {
3      std::cout << now_time << " : timer expired." << std::endl;
4   }
5
6   int main()
7   {
8      asio::io_service asio_service;
9
10     asio::deadline_timer timer( asio_service,
11                                 boost::posix_time::seconds(5) );
12
13
14     std::cout << now_time << " : request async_wait 5-seconds" << std::endl;
15     timer.async_wait( timer_expired );
16
17     std::cout << now_time << " : sleep 3-seconds" << std::endl;
18     boost::this_thread::sleep( boost::posix_time::seconds(3) );
19
20     std::cout << now_time << " : calling io_service::run" << std::endl;
```

Boost.Bind Introduction

```
int divide( int x, int y )
{
   return x / y;
}
```

```
int result = bind( divide, _1, _2 )( 10, 5 );
std::cout << "result: " << result << std::endl;
```

`result: 2`

```
boost::function<int(int,int)> func = bind( divide, _1, _2 );
int result = func( 10, 5 );
```

`result: 2`

```
int result = bind( divide, _2, _1 )( 10, 5 );
```

`result: 0`

```
int result = bind( divide, _1, 5 )( 10 );
```

`result: 2`

```
int result = bind( divide, 10, _1 )( 5 );
```

`result: 2`

```
int result = bind( divide, 20, _5 )( 1, 5, 9, 8, 2, 10 );
```

`result: 10`

Boost.Bind Introduction

```
int divide( int x, int y )
{
   return x / y;
}
```

```
int result = bind( divide, _1, _2 )( 10, 5 );
std::cout << "result: " << result << std::endl;
```

result: 2

```
boost::function<int(int,int)> func = bind( divide, _1, _2 );
int result = func( 10, 5 );
```

result: 2

```
int result = bind( divide, _2, _1 )( 10, 5 );
```

result: 0

```
int result = bind( divide, _1, 5 )( 10 );
```

result: 2

```
int result = bind( divide, 10, _1 )( 5 );
```

result: 2

```
int result = bind( divide, 20, _5 )( 1, 5, 9, 8, 2, 10 );
```

result: 10

Boost.Bind Introduction

```
int divide( int x, int y )
{
    return x / y;
}
```

```
int result = bind( divide, _1, _2 )( 10, 5 );
std::cout << "result: " << result << std::endl;
```

result: 2

```
boost::function<int(int,int)> func = bind( divide, _1, _2 );
int result = func( 10, 5 );
```

result: 2

```
int result = bind( divide, _2, _1 )( 10, 5 );
```

result: 0

```
int result = bind( divide, _1, 5 )( 10 );
```

result: 2

```
int result = bind( divide, 10, _1 )( 5 );
```

result: 2

```
int result = bind( divide, 20, _5 )( 1, 5, 9, 8, 2, 10 );
```

result: 10

Boost.Bind Introduction

```
int divide( int x, int y )
{
   return x / y;
}
```

```
int result = bind( divide, _1, _2 )( 10, 5 );
std::cout << "result: " << result << std::endl;
```

result: 2

```
boost::function<int(int,int)> func = bind( divide, _1, _2 );
int result = func( 10, 5 );
```

result: 2

```
int result = bind( divide, _2, _1 )( 10, 5 );
```

result: 0

```
int result = bind( divide, _1, 5 )( 10 );
```

result: 2

```
int result = bind( divide, 10, _1 )( 5 );
```

result: 2

```
int result = bind( divide, 20, _5 )( 1, 5, 9, 8, 2, 10 );
```

result: 10

Boost.Bind Introduction

```
int divide( int x, int y )
{
    return x / y;
}
```

```
int result = bind( divide, _1, _2 )( 10, 5 );
std::cout << "result: " << result << std::endl;
```

result: 2

```
boost::function<int(int,int)> func = bind( divide, _1, _2 );
int result = func( 10, 5 );
```

result: 2

```
int result = bind( divide, _2, _1 )( 10, 5 );
```

result: 0

```
int result = bind( divide, _1, 5 )( 10 );
```

result: 2

```
int result = bind( divide, 10, _1 )( 5 );
```

result: 2

```
int result = bind( divide, 20, _5 )( 1, 5, 9, 8, 2, 10 );
```

result: 10

Boost.Bind Introduction

```
int divide( int x, int y )
{
    return x / y;
}
```

```
int result = bind( divide, _1, _2 )( 10, 5 );
std::cout << "result: " << result << std::endl;
```

result: 2

```
boost::function<int(int,int)> func = bind( divide, _1, _2 );
int result = func( 10, 5 );
```

result: 2

```
int result = bind( divide, _2, _1 )( 10, 5 );
```

result: 0

```
int result = bind( divide, _1, 5 )( 10 );
```

result: 2

```
int result = bind( divide, 10, _1 )( 5 );
```

result: 2

```
int result = bind( divide, 20, _5 )( 1, 5, 9, 8, 2, 10 );
```

result: 10

Boost.Bind Introduction - Continued

```
int add( int x, int y )
{
    return x + y;
}
```

```
int x = 5;
boost::function<int()> func = bind( add, 20, x );
int result = func();
```

result: 25

```
int x = 5;
boost::function<int()> func = bind( add, 20, x );

x = 10;
int result = func();
```

result: 25

```
int x = 5;
boost::function<int()> func = bind( add, 20, boost::ref(x) );

x = 10;
int result = func();
```

result: 30

Boost.Bind Introduction - Continued

```
int add( int x, int y )
{
    return x + y;
}
```

```
int x = 5;
boost::function<int()> func = bind( add, 20, x );
int result = func();
```

result: 25

```
int x = 5;
boost::function<int()> func = bind( add, 20, x );

x = 10;
int result = func();
```

result: **25**

```
int x = 5;
boost::function<int()> func = bind( add, 20, boost::ref(x) );

x = 10;
int result = func();
```

result: 30

Boost.Bind Introduction - Continued

```
int add( int x, int y )
{
    return x + y;
}
```

```
int x = 5;
boost::function<int()> func = bind( add, 20, x );
int result = func();
```

result: 25

```
int x = 5;
boost::function<int()> func = bind( add, 20, x );

x = 10;
int result = func();
```

result: 25

```
int x = 5;
boost::function<int()> func = bind( add, 20, boost::ref(x) );

x = 10;
int result = func();
```

result: 30

Boost.Bind Introduction - Continued

```
class adder
{
   public:
      adder() : last_(0) {}
      int add(int x, int y){ last_ = x + y; return last_; }
      int operator()(int x, int y){ add(x,y); }
      int last(){ return last_; }
   private:
      int last_;
};
```

```
adder my_adder;
boost::function<int(int,int)> func = bind( &adder::add,
                                            my_adder,
                                            _1, _2 );

int result = func( 16, 8 );
```

```
result:  24
last:  0
```

Boost.Bind Introduction - Continued

```cpp
class adder
{
   public:
      adder() : last_(0) {}
      int add(int x, int y){ last_ = x + y; return last_; }
      int operator()(int x, int y){ add(x,y); }
      int last(){ return last_; }
   private:
      int last_;
};
```

```cpp
  adder my_adder;
  boost::function<int(int,int)> func = bind( &adder::add,
                                             boost::ref(my_adder),
                                             _1, _2 );

  int result = func( 16, 8 );
```

```
result:  24
last:  24
```

Boost.Bind Introduction - Continued

```
class adder
{
   public:
      adder() : last_(0) {}
      int add(int x, int y){ last_ = x + y; return last_; }
      int operator()(int x, int y){ add(x,y); }
      int last(){ return last_; }
   private:
      int last_;
};
```

```
   adder my_adder;
   boost::function<int(int,int)> func = bind( &adder::add,
                                              &my_adder,
                                              _1, _2 );

   int result = func( 16, 8 );
```

```
result:  24
last:  24
```

Binding Completion Handlers

```
1  void timer_expired( std::string identifier,
2                      const boost::system::error_code& e )
3  {
4     std::cout << identifier << " timer expired." << std::endl;
5  }
6
7  int main()
8  {
9     asio::io_service asio_service;
10
11    asio::deadline_timer timer1( asio_service,
12                                 boost::posix_time::seconds(5) );
13
14    asio::deadline_timer timer2( asio_service,
15                                 boost::posix_time::seconds(3) );
16
17
18    timer1.async_wait( boost::bind( timer_expired,
19                                    "timer1", _1 )       );
20
21    timer2.async_wait( boost::bind( timer_expired,
22                                    "timer2", _1 )       );
23
24    std::cout << "calling io_service::run" << std::endl;
25
26    asio_service.run();
27
28    std::cout << "done." << std::endl;
29    return 1;
30 }
```

Binding Completion Handlers

Output

```
calling io_service::run
timer2 timer expired.
timer1 timer expired.
done.
```

```
1   void timer_expired( std::string identifier,
2                       const boost::system::error_code& e )
3   {
4      std::cout << identifier << " timer expired." << std::endl;
5   }
6
7   int main()
8   {
9      asio::io_service asio_service;
10
11     asio::deadline_timer timer1( asio_service,
12                                  boost::posix_time::seconds(5) );
13
14     asio::deadline_timer timer2( asio_service,
15                                  boost::posix_time::seconds(3) );
16
17
18     timer1.async_wait( boost::bind( timer_expired,
19                                     "timer1", _1 )       );
20
21     timer2.async_wait( boost::bind( timer_expired,
```

Timer with Boost.Thread

```
1  void timer_expired( std::string identifier,
2                      const boost::system::error_code& e )
3  {
4    std::cout << now_time << " " << identifier << " timer expired enter." << std::endl;
5    boost::this_thread::sleep( boost::posix_time::seconds(3) );
6    std::cout << now_time << " " << identifier << " timer expired leave." << std::endl;
7  }
8
9  int main()
10 {
11    boost::asio::io_service asio_service;
12
13    boost::asio::deadline_timer timer1( asio_service,
14                                        boost::posix_time::seconds(5) );
15
16    boost::asio::deadline_timer timer2( asio_service,
17                                        boost::posix_time::seconds(5) );
18
19
20    timer1.async_wait( boost::bind( timer_expired,
21                                    "timer1", _1 )       );
22
23    timer2.async_wait( boost::bind( timer_expired,
24                                    "timer2", _1 )       );
25
26    boost::thread butler( boost::bind( &asio::io_service::run,
27                                       &asio_service ) );
28
29    butler.join();
30    std::cout << "done." << std::endl;
31
32    return 1;
33 }
```

Timer with Boost.Thread

Output

```
2010-May-08 20:26:44 timer1 timer expired enter.
2010-May-08 20:26:47 timer1 timer expired leave.
2010-May-08 20:26:47 timer2 timer expired enter.
2010-May-08 20:26:50 timer2 timer expired leave.
done.
```

The butler can only deliver one slushie at a time.

```
1   void timer_expired( std::string identifier,
2                       const boost::system::error_code& e )
3   {
4     std::cout << now_time << " " << identifier << " timer expired enter." << std::endl;
5     boost::this_thread::sleep( boost::posix_time::seconds(3) );
6     std::cout << now_time << " " << identifier << " timer expired leave." << std::endl;
7   }
8
9   int main()
10  {
11     boost::asio::io_service asio_service;
12
13     boost::asio::deadline_timer timer1( asio_service,
14                                         boost::posix_time::seconds(5) );
15
16     boost::asio::deadline_timer timer2( asio_service,
17                                         boost::posix_time::seconds(5) );
18
19
20     timer1.async_wait( boost::bind( timer_expired,
21                                     "timer1", _1 )       );
```

Timer with Two Boost.Threads

```
1  void timer_expired( std::string identifier, const boost::system::error_code& e )
2  {
3    std::cout << now_time << " " << identifier << " timer expired enter." << std::endl;
4    boost::this_thread::sleep( boost::posix_time::seconds(3) );
5    std::cout << now_time << " " << identifier << " timer expired leave." << std::endl;
6  }
7
8  int main()
9  {
10   boost::asio::io_service asio_service;
11
12   boost::asio::deadline_timer timer1( asio_service,
13                                       boost::posix_time::seconds(5) );
14
15   boost::asio::deadline_timer timer2( asio_service,
16                                       boost::posix_time::seconds(5) );
17
18   timer1.async_wait( boost::bind( timer_expired,
19                                   "timer1", _1 )      );
20
21   timer2.async_wait( boost::bind( timer_expired,
22                                   "timer2", _1 )      );
23
24   boost::thread_group thread_pool;
25
26   thread_pool.create_thread( boost::bind( &asio::io_service::run,
27                                           &asio_service ) );
28
29   thread_pool.create_thread( boost::bind( &asio::io_service::run,
30                                           &asio_service ) );
31
32   thread_pool.join_all();
33   std::cout << "done." << std::endl;
34   return 1;
35 }
```

Timer with Two Boost.Threads

Output

```
2010-May-08 20:33:49 timer1 timer expired enter.
2010-May-08 20:33:49 timer2 timer expired enter.
2010-May-08 20:33:52 timer1 timer expired leave.
2010-May-08 20:33:52 timer2 timer expired leave.
done.
```

```
1  void timer_expired( std::string identifier, const boost::system::error_code& e )
2  {
3    std::cout << now_time << " " << identifier << " timer expired enter." << std::endl;
4    boost::this_thread::sleep( boost::posix_time::seconds(3) );
5    std::cout << now_time << " " << identifier << " timer expired leave." << std::endl;
6  }
7
8  int main()
9  {
10    boost::asio::io_service asio_service;
11
12    boost::asio::deadline_timer timer1( asio_service,
13                                        boost::posix_time::seconds(5) );
14
15    boost::asio::deadline_timer timer2( asio_service,
16                                        boost::posix_time::seconds(5) );
17
18    timer1.async_wait( boost::bind( timer_expired,
19                                    "timer1", _1 )      );
20
21    timer2.async_wait( boost::bind( timer_expired,
22                                    "timer2", _1 )      );
23
24    boost::thread_group thread_pool;
```

Posting Work

Equivalent to the Owner placing items directly on the completion table.

```
void do_work( std::string work_value )
{
  std::cout << work_value << std::endl;
}

int main()
{
   boost::asio::io_service asio_service;

   asio_service.post( boost::bind( do_work, "eat" ) );
   asio_service.post( boost::bind( do_work, "drink" ) );
   asio_service.post( boost::bind( do_work, "and be merry!" ) );

   boost::thread butler( boost::bind( &asio::io_service::run,
                                      &asio_service ) );

   butler.join();
   std::cout << "done." << std::endl;

   return 1;
}
```

Posting Work

Output

```
eat
drink
and be merry!
done.
```

```
void do_work( std::string work_value )
{
  std::cout << work_value << std::endl;
}

int main()
{
   boost::asio::io_service asio_service;

   asio_service.post( boost::bind( do_work, "eat" ) );
   asio_service.post( boost::bind( do_work, "drink" ) );
   asio_service.post( boost::bind( do_work, "and be merry!" ) );

   boost::thread butler( boost::bind( &asio::io_service::run,
                                      &asio_service ) );

   butler.join();
   std::cout << "done." << std::endl;

   return 1;
}
```

Keeping the Butler Busy

```
void timer_expired( const boost::system::error_code& e )
{
  std::cout << now_time << " timer expired." << std::endl;
}

void do_work( int work_value )
{
  std::cout << now_time << " work " << work_value << std::endl;
}

int main()
{
   boost::asio::io_service asio_service;

   boost::asio::deadline_timer timer( asio_service,
                                      boost::posix_time::seconds(5) );

   timer.async_wait( boost::bind( timer_expired, _1 ) );

   boost::thread butler( boost::bind( &asio::io_service::run,
                                      &asio_service ) );

   for( int i=0; i<10; ++i )
   {
      asio_service.post( boost::bind( do_work, i ) );
      boost::this_thread::sleep( boost::posix_time::seconds(1) );
   }

   butler.join();
   std::cout << "done." << std::endl;

   return 1;
}
```

Keeping the Butler Busy

Output

```
2010-May-08 20:44:48 work 0
2010-May-08 20:44:49 work 1
2010-May-08 20:44:50 work 2
2010-May-08 20:44:51 work 3
2010-May-08 20:44:52 work 4
2010-May-08 20:44:53 timer expired.
done.
```

What happened to the rest of the work!

```
void timer_expired( const boost::system::error_code& e )
{
  std::cout << now_time << " timer expired." << std::endl;
}

void do_work( int work_value )
{
  std::cout << now_time << " work " << work_value << std::endl;
}

int main()
{
   boost::asio::io_service asio_service;

   boost::asio::deadline_timer timer( asio_service,
                                      boost::posix_time::seconds(5) );

   timer.async_wait( boost::bind( timer_expired, _1 ) );
```

The Work Object

```
void timer_expired( const boost::system::error_code& e )
{
  std::cout << now_time << " timer expired." << std::endl;
}

void do_work( int work_value )
{
  std::cout << now_time << " work " << work_value << std::endl;
}

int main()
{
   asio::io_service asio_service;

   asio::io_service::work* work = new asio::io_service::work( asio_service );

   asio::deadline_timer timer( asio_service,
                               boost::posix_time::seconds(5) );

   timer.async_wait( boost::bind( timer_expired, _1 ) );

   boost::thread butler( boost::bind( &asio::io_service::run, &asio_service ) );

   for( int i=0; i<10; ++i )
   {
      asio_service.post( boost::bind( do_work, i ) );
      boost::this_thread::sleep( boost::posix_time::seconds(1) );
   }

   delete work;

   butler.join();
   std::cout << "done." << std::endl;
   return 1;
}
```

The Work Object

Output

```
2010-May-08 20:51:01 work 0
2010-May-08 20:51:02 work 1
2010-May-08 20:51:03 work 2
2010-May-08 20:51:04 work 3
2010-May-08 20:51:05 work 4
2010-May-08 20:51:06 timer expired.
2010-May-08 20:51:06 work 5
2010-May-08 20:51:07 work 6
2010-May-08 20:51:08 work 7
2010-May-08 20:51:09 work 8
2010-May-08 20:51:10 work 9
done.
```

If we don't destroy the work object, the threads will never join.

```
1  void timer_expired( const boost::system::error_code& e )
2  {
3    std::cout << now_time << " timer expired." << std::endl;
4  }
5
6  void do_work( int work_value )
7  {
8    std::cout << now_time << " work " << work_value << std::endl;
9  }
10
11
```

What if Johnny Can't Handle Two Slushies

```
void timer_expired( std::string identifier, const boost::system::error_code& e )
{
  std::cout << now_time << " " << identifier << " timer expired enter." << std::endl;
  std::cout << now_time << " " << identifier << " timer expired leave." << std::endl;
}

int main()
{
   boost::asio::io_service asio_service;

   boost::asio::deadline_timer timer1( asio_service,
                                       boost::posix_time::seconds(5) );

   boost::asio::deadline_timer timer2( asio_service,
                                       boost::posix_time::seconds(5) );


   timer1.async_wait( boost::bind( timer_expired,
                                   "timer1", _1 )       );

   timer2.async_wait( boost::bind( timer_expired,
                                   "timer2", _1 )       );

   boost::thread_group thread_pool;

   thread_pool.create_thread( boost::bind( &asio::io_service::run, &asio_service ) );
   thread_pool.create_thread( boost::bind( &asio::io_service::run, &asio_service ) );

   thread_pool.join_all();
   std::cout << "done." << std::endl;
   return 1;
}
```

What if Johnny Can't Handle Two Slushies

Output

```
2010-May-08 15:38:24 timer2 timer expired
enter.2010-May-08 15:38:24
timer1 timer expired enter.
2010-May-08 15:38:24 timer1 timer expired leave.
2010-May-08 15:38:24 timer2 timer expired leave.
done.
```

```
void timer_expired( std::string identifier, const boost::system::error_code& e )
{
  std::cout << now_time << " " << identifier << " timer expired enter." << std::endl;
  std::cout << now_time << " " << identifier << " timer expired leave." << std::endl;
}

int main()
{
   boost::asio::io_service asio_service;

   boost::asio::deadline_timer timer1( asio_service,
                                       boost::posix_time::seconds(5) );

   boost::asio::deadline_timer timer2( asio_service,
                                       boost::posix_time::seconds(5) );


   timer1.async_wait( boost::bind( timer_expired,
                                   "timer1", _1 )       );

   timer2.async_wait( boost::bind( timer_expired,
                                   "timer2", _1 )       );
```

The io_service::strand

```
void timer_expired( std::string identifier, const boost::system::error_code& e )
{
  std::cout << now_time << " " << identifier << " timer expired enter." << std::endl;
  std::cout << now_time << " " << identifier << " timer expired leave." << std::endl;
}

int main()
{
   asio::io_service asio_service;

   asio::deadline_timer timer1( asio_service,
                                boost::posix_time::seconds(5) );

   asio::deadline_timer timer2( asio_service,
                                boost::posix_time::seconds(5) );

   asio::io_service::strand strand( asio_service );

   timer1.async_wait( strand.wrap( boost::bind( timer_expired,
                                                "timer1", _1 ) ) );

   timer2.async_wait( strand.wrap( boost::bind( timer_expired,
                                                "timer2", _1 ) ) );

   boost::thread_group thread_pool;
   thread_pool.create_thread( boost::bind( &asio::io_service::run, &asio_service ) );
   thread_pool.create_thread( boost::bind( &asio::io_service::run, &asio_service ) );

   thread_pool.join_all();
   std::cout << "done." << std::endl;
   return 1;
}
```

The io_service::strand

Output

```
2010-May-08 21:32:10 timer1 timer expired enter.
2010-May-08 21:32:10 timer1 timer expired leave.
2010-May-08 21:32:10 timer2 timer expired enter.
2010-May-08 21:32:10 timer2 timer expired leave.
done.
```

```
1   void timer_expired( std::string identifier, const boost::system::error_code& e )
2   {
3     std::cout << now_time << " " << identifier << " timer expired enter." << std::endl;
4     std::cout << now_time << " " << identifier << " timer expired leave." << std::endl;
5   }
6
7   int main()
8   {
9      asio::io_service asio_service;
10
11     asio::deadline_timer timer1( asio_service,
12                                  boost::posix_time::seconds(5) );
13
14     asio::deadline_timer timer2( asio_service,
15                                  boost::posix_time::seconds(5) );
16
17     asio::io_service::strand strand( asio_service );
18
19     timer1.async_wait( strand.wrap( boost::bind( timer_expired,
20                                                  "timer1", _1 ) ) );
21
22     timer2.async_wait( strand.wrap( boost::bind( timer_expired,
23                                                  "timer2", _1 ) ) );
24
```

shared_ptr Introduction

```
struct printer
{
   printer() { std::cout << "printer created" << std::endl; }
   ~printer() { std::cout << "printer destroyed" << std::endl; }

   void print( int x ){ std::cout << "printer(" << x << ")" << std::endl; }
};
```

```
   {
      std::cout << "entered scope" << std::endl;

      boost::shared_ptr< printer > my_printer( new printer );

      my_printer->print( 8 );

      std::cout << "leaving scope" << std::endl;
   }
   std::cout << "left scope" << std::endl;
```

Output

```
entered scope
printer created
printer(8)
leaving scope
printer destroyed
left scope
```

shared_ptr Introduction

```
struct printer
{
   printer() { std::cout << "printer created" << std::endl; }
   ~printer() { std::cout << "printer destroyed" << std::endl; }

   void print( int x ){ std::cout << "printer(" << x << ")" << std::endl; }
};
```

```
   {
      std::cout << "entered scope" << std::endl;

      boost::shared_ptr< printer > my_printer( new printer );

      my_printer->print( 8 );

      std::cout << "leaving scope" << std::endl;
   }
   std::cout << "left scope" << std::endl;
```

Output

```
entered scope
printer created
printer(8)
leaving scope
printer destroyed
left scope
```

shared_ptr Introduction – shared_from_this

```
struct shared_printer : boost::enable_shared_from_this< shared_printer >
{
   shared_printer() { std::cout << "shared_printer created" << std::endl; }
   ~shared_printer() { std::cout << "shared_printer destroyed" << std::endl; }
   void print( int x ){ std::cout << "shared_printer(" << x << ")" << std::endl; }

   boost::function<void(int)> get_printer()
   {
      return bind( &shared_printer::print,
                   shared_from_this(),
                   _1 );
   }
};
```

```
   {
      std::cout << "entered scope 1" << std::endl;
      boost::function< void(int) > print_func;
      {
         std::cout << "entered scope 2" << std::endl;

         boost::shared_ptr< shared_printer > my_printer( new shared_printer );

         my_printer->print( 8 );

         print_func = my_printer->get_printer();

         std::cout << "leaving scope 2" << std::endl;
      }
      std::cout << "left scope 2" << std::endl;

      print_func( 42 );
      std::cout << "leaving scope 1" << std::endl;
   }
   std::cout << "left scope 1" << std::endl;
```

Output

```
entered scope 1
entered scope 2
shared_printer created
shared_printer(8)
leaving scope 2
left scope 2
shared_printer(42)
leaving scope 1
shared_printer destroyed
left scope 1
```

```
struct shared_printer : boost::enable_shared_from_this< shared_printer >
{
   shared_printer() { std::cout << "shared_printer created" << std::endl; }
   ~shared_printer() { std::cout << "shared_printer destroyed" << std::endl; }
   void print( int x ){ std::cout << "shared_printer(" << x << ")" << std::endl; }

   boost::function<void(int)> get_printer()
   {
      return bind( &shared_printer::print,
                   shared_from_this(),
                   _1 );
   }
};
```

```
   {
```

Putting the Concepts Together

```
14  class typical_kid : public boost::enable_shared_from_this< typical_kid >
15  {
16     public:
17        typical_kid( asio::io_service& service ) : io_service_(service), strand_(service)
18        {  std::cout << now_time << " typical_kid created" << std::endl;    }
19
20        ~typical_kid()
21        {  std::cout << now_time << " typical_kid destroyed" << std::endl; }
22
23        void walk()
24        {
25           io_service_.post( strand_.wrap( boost::bind( &typical_kid::walk_impl,
26                                                          shared_from_this() )     ) );
27        }
28
29        void chew_gum( std::string flavor )
30        {
31           io_service_.post( strand_.wrap( boost::bind( &typical_kid::chew_gum_impl,
32                                                          shared_from_this(),  flavor ) ) );
33        }
34
35        void talk()
36        {
37           std::cout << now_time << " yackity yack yack." << std::endl;
38        }
39
40     private:

55        asio::io_service& io_service_;
56        asio::io_service::strand strand_;
57  };
```

Putting the Concepts Together

```
16      public:

23          void walk()
24          {
25              io_service_.post( strand_.wrap( boost::bind( &typical_kid::walk_impl,
26                                                            shared_from_this() )     ) );
27          }
28
29          void chew_gum( std::string flavor )
30          {
31              io_service_.post( strand_.wrap( boost::bind( &typical_kid::chew_gum_impl,
32                                                            shared_from_this(),  flavor ) ) );
33          }
34
35          void talk()
36          {
37              std::cout << now_time << " yackity yack yack." << std::endl;
38          }

40      private:
41          void walk_impl()
42          {
43              std::cout << now_time << " ++++ start walking ++++" << std::endl;
44              boost::this_thread::sleep( boost::posix_time::seconds(3) );
45              std::cout << now_time << " ++++ done walking. ++++" << std::endl;
46          }
47
48          void chew_gum_impl( std::string flavor )
49          {
50              std::cout << now_time << " ---- start chewing " << flavor << " gum ----" << std:::e
51              boost::this_thread::sleep( boost::posix_time::seconds(2) );
52              std::cout << now_time << " ---- done chewing gum. ----" << std::endl;
53          }
```

Putting the Concepts Together

```
61  int main()
62  {
63     asio::io_service asio_service;
64     boost::shared_ptr<asio::io_service::work>
65                      work( new asio::io_service::work(asio_service) );
66
67     boost::thread_group butler_pool;
68     butler_pool.create_thread( boost::bind( &asio::io_service::run, &asio_service ) );
69     butler_pool.create_thread( boost::bind( &asio::io_service::run, &asio_service ) );
70     butler_pool.create_thread( boost::bind( &asio::io_service::run, &asio_service ) );
71
72     {
73        boost::shared_ptr<typical_kid> johnny( new typical_kid(asio_service) );
74
75        for( int i=0; i<10; ++i )
76        {
77           johnny->talk();
78           johnny->walk();
79           johnny->talk();
80           johnny->chew_gum("bubble");
81           johnny->talk();
82
83           boost::this_thread::sleep( boost::posix_time::seconds(3) );
84        }
85
86        std::cout << now_time << " leaving the beach......." << std::endl;
87     }
88
89     work.reset();
90
91     butler_pool.join_all();
92     std::cout << now_time << " done." << std::endl;
93     return 1;
94  }
```

Putting the Concepts Together

Output

```
2010-May-08 18:36:36 typical_kid created
2010-May-08 18:36:36 yackity yack yack.
2010-May-08 18:36:36 ++++ start walking ++++
2010-May-08 18:36:36 yackity yack yack.
2010-May-08 18:36:36 yackity yack yack.
2010-May-08 18:36:39 ++++ done walking.  ++++
2010-May-08 18:36:39 --- start chewing bubble gum ---
2010-May-08 18:36:39 yackity yack yack.
2010-May-08 18:36:39 yackity yack yack.
2010-May-08 18:36:39 yackity yack yack.
2010-May-08 18:36:41 --- done chewing gum.  ---
2010-May-08 18:36:41 ++++ start walking ++++
2010-May-08 18:36:42 yackity yack yack.
2010-May-08 18:36:42 yackity yack yack.
2010-May-08 18:36:42 yackity yack yack.
2010-May-08 18:36:44 ++++ done walking.  ++++
2010-May-08 18:36:44 --- start chewing bubble gum ---
2010-May-08 18:36:45 yackity yack yack.
2010-May-08 18:36:45 yackity yack yack.
2010-May-08 18:36:45 yackity yack yack.
2010-May-08 18:36:46 --- done chewing gum.  ---
```

```
61 int main()
62 {
63    asio::io_service asio_service;
64    boost::shared_ptr<asio::io_service::work>
65                work( new asio::io_service::work(asio_service) );
66
67    boost::thread_group butler_pool;
68    butler_pool.create_thread( boost::bind( &asio::io_service::run, &asio_service ) );
69    butler_pool.create_thread( boost::bind( &asio::io_service::run, &asio_service ) );
```

Outline

1. Introducing ASIO
   - Asynchronous I/O
   - Asio Basics
2. Communication with ASIO
   - Buffers
   - API
3. Flash XML Server
   - The Goal
   - The Server Class

Buffers

Asio deals with memory using *buffers*.

```
typdef std::pair<void*, std::size_t> mutable_buffer;
typdef std::pair<const void*, std::size_t> const_buffer;
```

mutable_buffer → const_buffer

Asio supports scatter/gather when buffers are stored in containers.

Buffers do not own the underlying data!

Buffers

Asio deals with memory using *buffers*.

```
typdef std::pair<void*, std::size_t> mutable_buffer;
typdef std::pair<const void*, std::size_t> const_buffer;
```

mutable_buffer → const_buffer

Asio supports scatter/gather when buffers are stored in containers.

Buffers do not own the underlying data!

Buffers

Asio deals with memory using *buffers*.

```
class mutable_buffer;
class const_buffer;
```

mutable_buffer $\longrightarrow$ const_buffer

Asio supports scatter/gather when buffers are stored in containers.

Buffers do not own the underlying data!

Buffers

Asio deals with memory using *buffers*.

```
class mutable_buffer;
class const_buffer;
```

mutable_buffer → const_buffer

Asio supports scatter/gather when buffers are stored in containers.

Buffers do not own the underlying data!

Buffers

Asio deals with memory using *buffers*.

```
class mutable_buffer;
class const_buffer;
```

mutable_buffer → const_buffer

Asio supports scatter/gather when buffers are stored in containers.

Buffers do not own the underlying data!

Buffers - Continued

It is easy to get an Asio buffer.

use

```
boost::asio::buffer(...)
```

```
1 socket_.send( asio::buffer(data, size) );
```

```
1 std::string personal_message( "dinner time!" );
2 socket_.send( asio::buffer(personal_message) );
```

```
1 boost::array<uint_8,4> code = { 0xde, 0xad, 0xbe, 0xef };
2 socket_.send( asio::buffer(code) );
```

Outline

1. Introducing ASIO
   - Asynchronous I/O
   - Asio Basics
2. Communication with ASIO
   - Buffers
   - API
3. Flash XML Server
   - The Goal
   - The Server Class

Sending and Receiving

```
asio::async_read
asio::async_read_until
asio::async_write
```

```
asio::ip::tcp::socket::async_read_some
```

```
asio::ip::tcp::socket::async_write_some
```

Outline

1 Introducing ASIO
- Asynchronous I/O
- Asio Basics

2 Communication with ASIO
- Buffers
- API

3 Flash XML Server
- The Goal
- The Server Class

Top Level Interaction

```
typedef boost::shared_ptr< FlashXMLHandler > xml_handler_t;
```

```
   asio_generic_server< FlashXMLHandler > xml_server( 1 );
   xml_server.start_server( 8989 );
   xml_server.add_connection_handler( &xml_client_connection );
```

```
void xml_client_connection( xml_handler_t handler )
{
   ...
   handler->add_message_handler( &xml_client_message );
}
```

```
void xml_client_message( xml_handler_t client, omd::XMLNode node )
{
   // ... something cool here
}
```

Outline

1. Introducing ASIO
   - Asynchronous I/O
   - Asio Basics
2. Communication with ASIO
   - Buffers
   - API
3. Flash XML Server
   - The Goal
   - The Server Class

Server class handles incoming connections

```
1  template <typename AsioConnectionHandler>
2  class asio_generic_server
3  {
4     public:
5        typedef boost::signal< void(boost::shared_ptr<AsioConnectionHandler>) > signal_type;
6        typedef typename signal_type::slot_type observer_type;
7
8     public:
9        asio_generic_server();
10
11       asio_generic_server( int thread_count_ );
12
13       ~asio_generic_server();
14
15       void start_server( int port );
16
17       void stop_server();
18
19    private:
20
21       void handle_new_connection( boost::shared_ptr<AsioConnectionHandler> handler,
22                                   const boost::system::error_code& error );
23
24       int thread_count;
25       boost::thread_group thread_pool;
26       asio::io_service asio_io_service;
27       asio::ip::tcp::acceptor acceptor;
28       signal_type connection_handler;
29 };
```

asio_generic_server::start_sever

```
19   void start_server( int port )
20   {
21     // get a new object that will handle our client interaction
22     boost::shared_ptr<AsioConnectionHandler>
23          handler(new AsioConnectionHandler(asio_io_service));
24
25     // set up the acceptor to listen on the tcp port
26     asio::ip::tcp::endpoint endpoint( asio::ip::tcp::v4(), port );
27     acceptor.open( endpoint.protocol() );
28     acceptor.set_option( asio::ip::tcp::acceptor::reuse_address( true ) );
29     acceptor.bind( endpoint );
30     acceptor.listen();
31
32     // request an asynchronous accept (listen)
33     acceptor.async_accept
34       ( handler->socket(),
35         boost::bind( &asio_generic_server<AsioConnectionHandler>::handle_new_connection,
36                      this,
37                      handler,
38                      asio::placeholders::error ) );
39
40
41     // start pool of threads to process the asio events
42     for( int i=0; i<thread_count; ++i )
43     {
44       thread_pool.create_thread( boost::bind( &asio::io_service::run,
45                                               &asio_io_service ) );
46     }
47   }
```

asio_generic_server::handle_new_connectio

```
59    void handle_new_connection( boost::shared_ptr<AsioConnectionHandler> handler,
60                                const boost::system::error_code& error )
61    {
62      if( !error )
63      {
64        handler->start();
65        connection_handler( handler );
66
67        boost::shared_ptr<AsioConnectionHandler>
68             new_handler( new AsioConnectionHandler( asio_io_service ) );
69
70        acceptor.async_accept
71          ( new_handler->socket(),
72            boost::bind( &asio_generic_server<AsioConnectionHandler>::handle_new_connection,
73                         this,
74                         new_handler,
75                         asio::placeholders::error ) );
76      }
77    }
```

FlashXMLHandler

```
class FlashXMLHandler
        : public boost::enable_shared_from_this< FlashXMLHandler >
{
   public:
      typedef boost::signal< void( boost::shared_ptr<FlashXMLHandler>,
                                   omd::XMLNode ) > signal_type;
      typedef signal_type::slot_type observer_type;

   public:
      FlashXMLHandler( boost::asio::io_service& service );
      ~FlashXMLHandler();

      void start();
      void send( const std::string& message );
      void send( omd::XMLNode node );
      boost::signals::connection add_message_handler( const observer_type& observer );
      boost::asio::ip::tcp::socket& socket();

   private:
      void read_packet();
      void read_packet_done( const boost::system::error_code&, int bytes_transferred );

      void queue_message( std::string message );
      void start_packet_send();
      void packet_send_done( const boost::system::error_code& error );

      asio::io_service& service_;
      asio::ip::tcp::socket socket_;
      asio::io_service::strand write_strand;
      asio::streambuf in_packet;
      std::deque<std::string> send_packet_queue;

      signal_type update_handler;
};
```

Private Members

```
27         asio::io_service& service_;
28         asio::ip::tcp::socket socket_;
29         asio::io_service::strand write_strand;

33         signal_type update_handler;
```

Constructor

```
1  FlashXMLHandler::FlashXMLHandler( asio::io_service& service )
2     : service_( service ),
3       socket_( service ),
4       write_strand( service)
5  {}
```

```
27 boost::signals::connection
28 FlashXMLHandler::add_message_handler( const observer_type& observer )
29 {
30    return update_handler.connect( observer );
31 }

33 boost::asio::ip::tcp::socket& FlashXMLHandler::socket()
34 {
35    return socket_;
36 }
```

Private Members

```
27        asio::io_service& service_;
28        asio::ip::tcp::socket socket_;
29        asio::io_service::strand write_strand;

33        signal_type update_handler;
```

Constructor

```
1  FlashXMLHandler::FlashXMLHandler( asio::io_service& service )
2     : service_( service ),
3       socket_( service ),
4       write_strand( service)
5  {}
```

```
27  boost::signals::connection
28  FlashXMLHandler::add_message_handler( const observer_type& observer )
29  {
30     return update_handler.connect( observer );
31  }
```

```
33  boost::asio::ip::tcp::socket& FlashXMLHandler::socket()
34  {
35     return socket_;
36  }
```

Server Starting Handler

asio_generic_server

```
59    void handle_new_connection( boost::shared_ptr<AsioConnectionHandler> handler,
60                                const boost::system::error_code& error )
61    {
62      if( !error )
63      {
64        handler->start();
```

Handler

```
7   void FlashXMLHandler::start()
8   {
9       read_packet();
10  }
```

```
38  void FlashXMLHandler::read_packet()
39  {
40      // read the packet header
41      asio::async_read_until( socket_,
42                              in_packet,
43                              '\0',        // stream is deliminated with a null
44                              boost::bind( &FlashXMLHandler::read_packet_done,
45                                           shared_from_this(),
46                                           asio::placeholders::error,
47                                           asio::placeholders::bytes_transferred ) );
48  }
```

Caution!

`async_read_until` may have read data into the buffer after the *until*

Server Starting Handler

`asio_generic_server`

```
59    void handle_new_connection( boost::shared_ptr<AsioConnectionHandler> handler,
60                                const boost::system::error_code& error )
61    {
62      if( !error )
63      {
64        handler->start();
```

Handler

```
7   void FlashXMLHandler::start()
8   {
9       read_packet();
10  }
```

```
38  void FlashXMLHandler::read_packet()
39  {
40      // read the packet header
41      asio::async_read_until( socket_,
42                              in_packet,
43                              '\0',       // stream is deliminated with a null
44                              boost::bind( &FlashXMLHandler::read_packet_done,
45                                           shared_from_this(),
46                                           asio::placeholders::error,
47                                           asio::placeholders::bytes_transferred ) );
48  }
```

Caution!

`async_read_until` may have read data into the buffer after the *until*

```
50  void FlashXMLHandler::read_packet_done( const boost::system::error_code& error,
51                                          int bytes_transferred )
52  {
53     if( error )
54     {
55        // ERROR occurred... do something clever
56        return;
57     }
58
59     omd::XMLNode packet;
60     omd::xml::parse( in_packet, packet );
61
62     // check if we received a policy file
63     if( packet.node_name() == "policy-file-request" )
64     {
65
66        std::string xml_packet
67              ( "<cross-domain-policy>"
68                "<allow-access-from domain='*' to-ports='*' secure='false' />"
69                "</cross-domain-policy>" );
70        send(xml_packet);
71     }
72     else
73     {
74        try
75        {
76           // update all of the listeners
77           update_handler( shared_from_this(), packet );
78        }
79        catch(...)
80        {}
81     }
82
83     // start the process over
84     read_packet();
85  }
```

Sending - Internal

```
20 void FlashXMLHandler::send( omd::XMLNode node )
21 {
22    std::stringstream stream;
23    stream << node;
24    send( stream.str() );
25 }
```

```
12 void FlashXMLHandler::send( const std::string& message )
13 {
14    service_.post
15       ( write_strand.wrap( boost::bind( &FlashXMLHandler::queue_message,
16                                         shared_from_this(),
17                                         message ) ) );
18 }
```

```
87 void FlashXMLHandler::queue_message( std::string message )
88 {
89    bool write_in_progress = !send_packet_queue.empty();
90    send_packet_queue.push_back( message );
91
92    // if we aren't currently doing a write start one
93    if( !write_in_progress )
94    {
95       start_packet_send();
96    }
97 }
```

Sending - Internal

```
20 void FlashXMLHandler::send( omd::XMLNode node )
21 {
22    std::stringstream stream;
23    stream << node;
24    send( stream.str() );
25 }
```

```
12 void FlashXMLHandler::send( const std::string& message )
13 {
14    service_.post
15       ( write_strand.wrap( boost::bind( &FlashXMLHandler::queue_message,
16                                         shared_from_this(),
17                                         message ) ) );
18 }
```

```
87 void FlashXMLHandler::queue_message( std::string message )
88 {
89    bool write_in_progress = !send_packet_queue.empty();
90    send_packet_queue.push_back( message );
91
92    // if we aren't currently doing a write start one
93    if( !write_in_progress )
94    {
95       start_packet_send();
96    }
97 }
```

Sending - External

```
99  void FlashXMLHandler::start_packet_send()
100 {
101    send_packet_queue.front() += '\0';
102
103    // register the send
104    asio::async_write
105        ( socket_,
106          asio::buffer( send_packet_queue.front() ),
107          write_strand.wrap( boost::bind( &FlashXMLHandler::packet_send_done,
108                                          shared_from_this(),
109                                          asio::placeholders::error ) )
110          );
111 }

113 void FlashXMLHandler::packet_send_done( const boost::system::error_code& error )
114 {
115    if( !error )
116    {
117       // pop the sent packet from the deque and start the next one if we have more
118       send_packet_queue.pop_front();
119
120       if( !send_packet_queue.empty() )
121       {
122          start_packet_send();
123       }
124    }
125    else
126    { // ERROR occurred... do something clever   }
127 }
```

Sending - External

```
99   void FlashXMLHandler::start_packet_send()
100  {
101     send_packet_queue.front() += '\0';
102
103     // register the send
104     asio::async_write
105        ( socket_,
106          asio::buffer( send_packet_queue.front() ),
107          write_strand.wrap( boost::bind( &FlashXMLHandler::packet_send_done,
108                                          shared_from_this(),
109                                          asio::placeholders::error ) )
110           );
111  }
```

```
113  void FlashXMLHandler::packet_send_done( const boost::system::error_code& error )
114  {
115     if( !error )
116     {
117        // pop the sent packet from the deque and start the next one if we have more
118        send_packet_queue.pop_front();
119
120        if( !send_packet_queue.empty() )
121        {
122           start_packet_send();
123        }
124     }
125     else
126     { // ERROR occurred... do something clever    }
127  }
```

Thoughts...

Decouple I/O from processing

```
1  void xml_client_message( xml_handler_t client, omd::XMLNode node )
2  {
3     processing_service_.post( boost::bind( process_incoming_message,
4                                            client,
5                                            node ) );
6  }
```

Use `boost::signals2`

Use strands!

Thoughts...

Decouple I/O from processing

```
1  void xml_client_message( xml_handler_t client, omd::XMLNode node )
2  {
3     processing_service_.post( boost::bind( process_incoming_message,
4                                            client,
5                                            node ) );
6  }
```

Use boost::signals2

Use strands!

Thoughts...

Decouple I/O from processing

```
1  void xml_client_message( xml_handler_t client, omd::XMLNode node )
2  {
3     processing_service_.post( boost::bind( process_incoming_message,
4                                             client,
5                                             node ) );
6  }
```

Use `boost::signals2`

Use strands!