Managing code complexity in asynchronous, distributed server architectures

Karl Berg

Senior Systems Engineer, Piranha Games Inc.

GAME DEVELOPERS CONFERENCE 2014
SAN FRANCISCO, CA
MARCH 17-21, 2014
EXPO DATES: MARCH 19-21

Background

- Networking and client-server architecture
- Serialization
- Threading
- C++ for example code

Problem Domain

- Two approaches

Problem Domain

- Blocking model
  - Massively threaded

Problem Domain

- Blocking model
  - Massively threaded
  - One thread dedicated per request

Problem Domain

- Blocking model
  - Massively threaded
  - One thread dedicated per request
  - Blocking

Problem Domain

- Blocking model
    - Massively threaded
    - One thread dedicated per request
    - Blocking
    - Easy to maintain!

Problem Domain

- Blocking model

APACHE
HTTP SERVER

MySQL®

Problem Domain

- Problems with blocking?

Problem Domain

- Problems with blocking?
  - Peak Concurrent Users

Problem Domain

- Problems with blocking?
  - Peak Concurrent Users
  - Massively threaded = high overhead

Problem Domain

- Problems with blocking?
  - Peak Concurrent Users
  - Massively threaded = high overhead
  - Memory

Problem Domain

- Problems with blocking?
  - Peak Concurrent Users
  - Massively threaded = high overhead
  - Memory
  - CPU

Problem Domain

- Event driven model

Problem Domain

- Event driven model
  - One thread per core

Problem Domain

- Event driven model
  - One thread per core
  - Stateless

Problem Domain

- Event driven model
  - One thread per core
  - Stateless
  - Distributed

Problem Domain

- Event driven model
  - One thread per core
  - Stateless
  - Distributed
  - Asynchronous

Problem Domain

- Event driven model
  - IOCP
  - kqueue
  - epoll

Problem Domain

- Event driven model

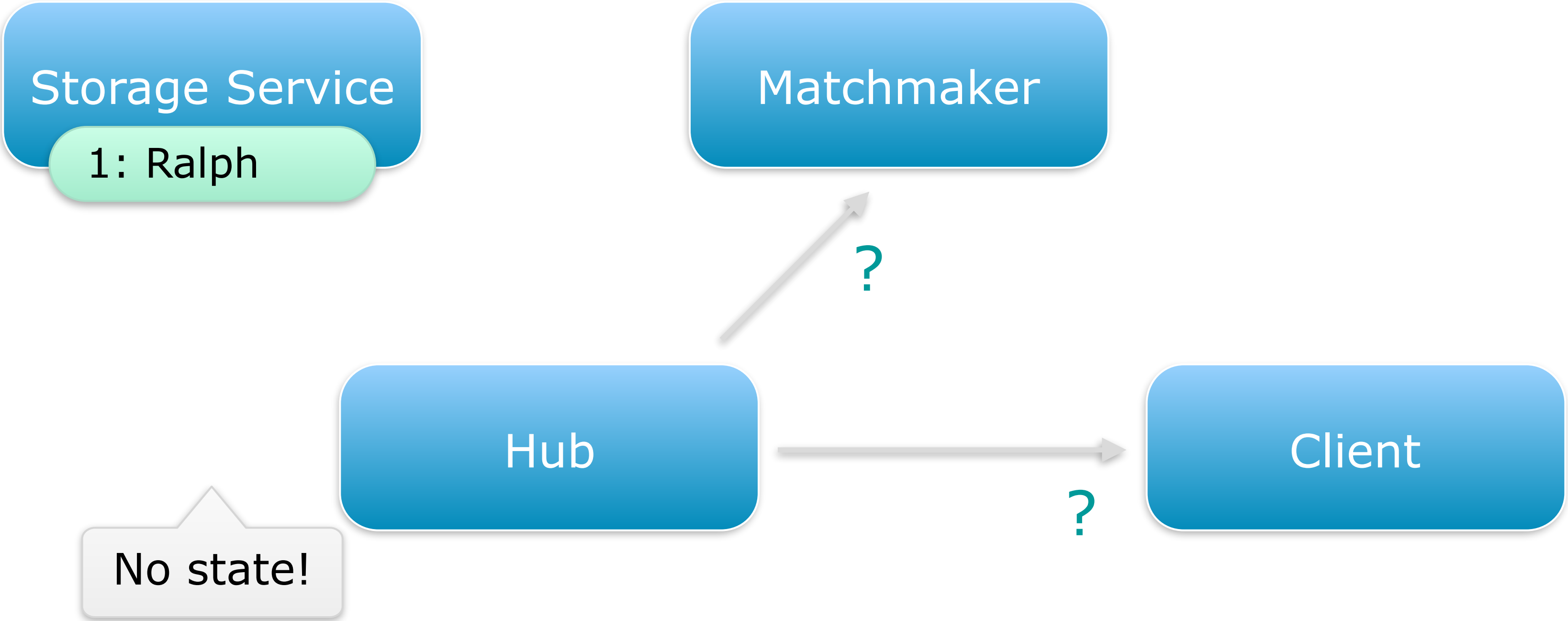
Problem Domain

- Problems with event driven?

Problem Domain

- Problems with event driven?
  - No state!

Problem Domain



Problem Domain

- Problems with event driven?
  - No state!
  - Broken up code

Problem Domain

- Problems with event driven?
  - No state!
  - Broken up code
  - Complicated error handling

Topics

Topics

- Automatic programming

Topics

- Automatic programming
  - Code auto-generation

Topics

- Automatic programming
    - Code auto-generation
    - Why use it

Topics

- Automatic programming
  - Code auto-generation
  - Why use it
  - Approaches for implementation

Topics

- Automatic programming
  - Code auto-generation
  - Why use it
  - Approaches for implementation
  - Best practices

Topics

- Defining a request or packet interface

Topics

- Defining a request or packet interface
  - Leverages automatic programming

Topics

- Defining a request or packet interface
    - Leverages automatic programming
    - Sets a baseline for additional topics

Topics

- Safely and efficiently managing state

Topics

- Safely and efficiently managing state
    - Some requests require state

Topics

- Safely and efficiently managing state
  - Some requests require state
  - Efficiency gains for distributed problems

Topics

- Safely and efficiently managing state
  - Some requests require state
  - Efficiency gains for distributed problems
  - Foundation for final topic

Topics

- Coroutines

Topics

- Coroutines
  - What are they?

Topics

- Coroutines
  - What are they?
  - Approaches for implementation

Topics

- Coroutines
  - What are they?
  - Approaches for implementation
  - How to make them safe

Why C++?

Why C++?

- Design and Team constraints

Why C++?

- Design and Team constraints
  - Client using CryEngine 3

Why C++?

- Design and Team constraints
  - Client using CryEngine 3
  - Design called for complicated, shared logic

Why C++?

- Design and Team constraints
  - Client using CryEngine 3
  - Design called for complicated, shared logic
  - No desire to duplicate code

Why C++?

- Design and Team constraints
  - Overwhelmingly C++ programmers

Why C++?

- Design and Team constraints
    - Overwhelmingly C++ programmers
    - Minimize ramp time for engineers

Why C++?

- Benefits

Why C++?

- Benefits
  - Shared library for common code and types

Why C++?

- Benefits
  - Shared library for common code and types
  - Robust ecosystem of libraries

Why C++?

- Benefits
  - Shared library for common code and types
  - Robust ecosystem of libraries
  - Minimal ramp time for engineers

Why C++?

- Drawbacks
  - Minimal support for asynchronous operations

Why C++?

- Drawbacks
  - Minimal support for asynchronous operations
  - Minimal support for robust threading

Why C++?

- Drawbacks
  - Minimal support for asynchronous operations
  - Minimal support for robust threading
  - Provides no stability/uptime guarantees

Automatic Programming

Automatic Programming

- What is it?

Automatic Programming

- What is it?
    - Make your compiler do the work

Automatic Programming

- What is it?
  - Make your compiler do the work
  - A form of code compression

Automatic Programming

- What is it?
  - Make your compiler do the work
  - A form of code compression
  - Can be cleanly integrated into your build

Autogenerating Code

- Why use it?

Autogenerating Code

- Why use it?
  - Provides an enormous productivity boost

Autogenerating Code

- Why use it?
  - Provides an enormous productivity boost
  - MWO: *10x* compression of server code!

Autogenerating Code

- Why use it?
  - Provides an enormous productivity boost
  - MWO: *10x* compression of server code!
  - 100k lines expands to ~1 million lines of C++

Autogenerating Code

- Why use it?
  - Can express complex repetitive actions

Autogenerating Code

- Why use it?
  - Can express complex repetitive actions
  - Handles cases that templates can’t

Autogenerating Code

- Why use it?
  - Can express complex repetitive actions
  - Handles cases that templates can’t
  - Data-driven approach

Components of Autogeneration

Components of Autogeneration

- Data files

Components of Autogeneration

- Data files
- Template files

Components of Autogeneration

- Data files
- Template files
- Definition files

Components of Autogeneration

- Data files

Components of Autogeneration

- Data files
  - Hierarchical

Components of Autogeneration

- Data files
  - Hierarchical
  - Should be easy to read and extend

Components of Autogeneration

- Data files
  - Hierarchical
  - Should be easy to read and extend
  - XML works well!

Components of Autogeneration

- Template files

Components of Autogeneration

- Template files
    - Transform data into code

Components of Autogeneration

- Template files
  - Transform data into code
  - Strong at string manipulation

Components of Autogeneration

- Template files
  - Transform data into code
  - Strong at string manipulation
  - Dedicated tools exist

django

Jinja

Components of Autogeneration

- Template files
  - Transform data into code
  - Strong at string manipulation
  - Dedicated tools exist
  - Write a custom language

Components of Autogeneration

- Template files
  - Transform data into code
  - Strong at string manipulation
  - Dedicated tools exist
  - Write a custom language
  - Use an existing script language

django

Jinja

perl

php

python

Components of Autogeneration

- Definition files

Components of Autogeneration

- Definition files
  - Driver for actual code expansion

Components of Autogeneration

- Definition files
  - Driver for actual code expansion
  - Define pairs of data and template inputs

Components of Autogeneration

- Definition files
    - Driver for actual code expansion
    - Define pairs of data and template inputs
    - May specify output filenames

Implementing Autogeneration

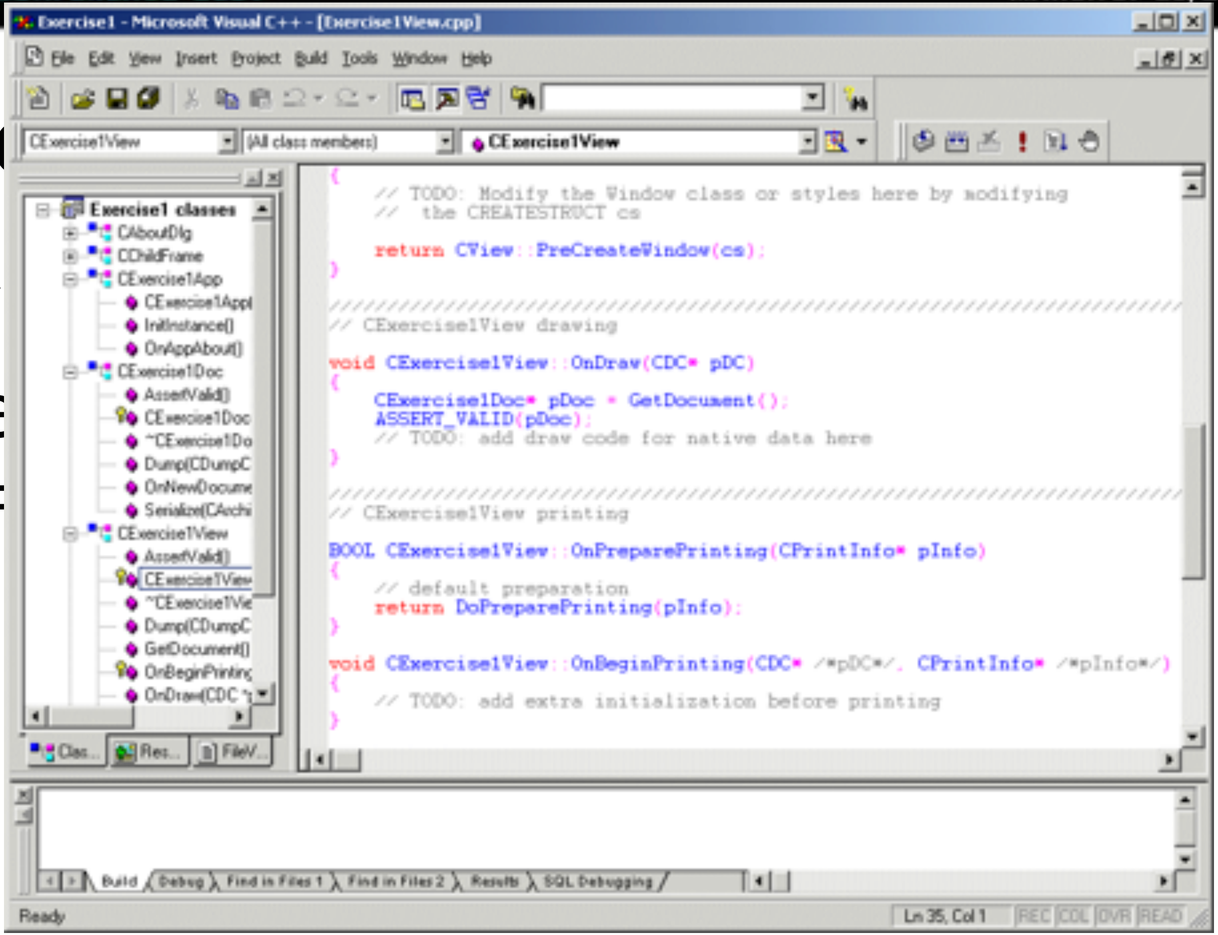
- Many valid approaches

Implementing Autogeneration

- Many valid approaches
- Some don’t work very well

Imple

- Many
- Some
    - MF



```
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CExercise1View drawing

void CExercise1View::OnDraw(CDC* pDC)
{
    CExercise1Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
}

/////////////////////////////////////////////////////////////////////////////
// CExercise1View printing

BOOL CExercise1View::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CExercise1View::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}
```

Implementing Autogeneration

- Many valid approaches
- Some don’t work very well
  - MFC/Visual C++ related trauma
  - Valuable lesson to be learned

Implementing Autogeneration

- Many valid approaches
- Some don’t work very well
  - MFC/Visual C++ related trauma
  - Valuable lesson to be learned
  - Never hand-edit autogenerated code!

Implementing Autogeneration

- MWO approach

Implementing Autogeneration

- MWO approach
  - Run autogeneration as pre-compile step

Implementing Autogeneration

- MWO approach
  - Run autogeneration as pre-compile step
  - Hand edits will be overwritten

Implementing Autogeneration

- MWO approach
  - Run autogeneration as pre-compile step
  - Hand edits will be overwritten
  - Forces devs to change autogen input files

Implementing Autogeneration

- MWO approach
  - Run autogeneration as pre-compile step
  - Hand edits will be overwritten
  - Forces devs to change autogen input files
  - Can inherit and extend

Implementing Autogeneration

- MWO approach
  - Run autogeneration as pre-compile step
  - Hand edits will be overwritten
  - Forces devs to change autogen input files
  - Can inherit and extend
  - Embed autogen output into project

Implementing Autogeneration

- You broke my compile times?!

Implementing Autogeneration

- You broke my compile times?!
    - Autogenerated output gets very big

Implementing Autogeneration

- You broke my compile times?!
  - Autogenerated output gets very big
  - Helps to have a set of guidelines

Implementing Autogeneration

- You broke my compile times?!
  - Autogenerated output gets very big
  - Helps to have a set of guidelines
  - Only autogenerate code if you need to

Implementing Autogeneration

- You broke my compile times?!
  - Autogenerated output gets very big
  - Helps to have a set of guidelines
  - Only autogenerate code if you need to
  - Only using an interface?

Implementing Autogeneration

- You broke my compile times?!
  - Autogenerated output gets very big
  - Helps to have a set of guidelines
  - Only autogenerate code if you need to
  - Only using an interface?
  - Try using a C++ template function

Implementing Autogeneration

- You broke my compile times?!
  - Manage your timestamps

Implementing Autogeneration

- You broke my compile times?!
    - Manage your timestamps
    - Want to avoid needless recompiles

Implementing Autogeneration

- You broke my compile times?!
  - Manage your timestamps
  - Want to avoid needless recompiles
  - Compiler can’t see autogen file dependencies

Implementing Autogeneration

- You broke my compile times?!
  - Manage your timestamps
  - Want to avoid needless recompiles
  - Compiler can’t see autogen file dependencies
  - Pre-build autogen can break iterative builds

Implementing Autogeneration

- You broke my compile times?!
  - Manage your timestamps
  - Want to avoid needless recompiles
  - Compiler can’t see autogen file dependencies
  - Pre-build autogen can break iterative builds
  - MWO autogeneration caches output and diffs

Best Practices with Autogen

- Compile-time asserts

Best Practices with Autogen

- Compile-time asserts
  - You WANT to fail at compile time

Best Practices with Autogen

- Compile-time asserts
  - You WANT to fail at compile time
  - C++11, Boost StaticAssert

Best Practices with Autogen

- Compile-time asserts
  - You WANT to fail at compile time
  - C++11, Boost StaticAssert
  - Can build your own using trickery

Best Practices with Autogen

- Compile-time metaprogramming

Best Practices with Autogen

- Compile-time metaprogramming
  - Combining templates and enums

Best Practices with Autogen

- Compile-time metaprogramming
  - Combining templates and enums
  - Outputs extremely efficient code

Best Practices with Autogen

- Compile-time metaprogramming
  - Combining templates and enums
  - Outputs extremely efficient code
  - Fails at compile time, this is good!

Best Practices with Autogen

- Compile-time metaprogramming
  - Combining templates and enums
  - Outputs extremely efficient code
  - Fails at compile time, this is good!
  - Can be difficult to understand

Best Practices with Autogen

- Avoiding name collisions

Best Practices with Autogen

- Avoiding name collisions
  - Can easily autogenerate name collisions

Best Practices with Autogen

- Avoiding name collisions
  - Can easily autogenerate name collisions
  - Two approaches for avoiding collisions

Best Practices with Autogen

- Avoiding name collisions
  - Can easily autogenerate name collisions
  - Two approaches for avoiding collisions
  - Namespaces and classes/structs

Best Practices with Autogen

- Avoiding name collisions
  - Can easily autogenerate name collisions
  - Two approaches for avoiding collisions
  - Namespaces and classes/structs
  - Understand when to use each

Best Practices with Autogen

- Structures are valid parameters for templates

```
struct test
{
};

template <typename T>
void function();
```

Best Practices with Autogen

- Structures are valid parameters for templates

```
struct test
{
};

template <typename T>
void function();

// This works!
function<test>();
```

Best Practices with Autogen

- Structures are valid parameters for templates
- Namespaces are not

```
struct test
{
};

template <typename T>
void function();

// This works!
function<test>();
```

```
namespace test
{
}

template <typename T>
void function();
```

Best Practices with Autogen

- Structures are valid parameters for templates
- Namespaces are not

```
struct test
{
};

template <typename T>
void function();

// This works!
function<test>();
```

```
namespace test
{
}

template <typename T>
void function();


function<test>();
```

Best Practices with Autogen

- Structures are valid parameters for templates
- Namespaces are not

```
struct test
{
};

template <typename T>
void function();

// This works!
function<test>();
```

```
namespace test
{
}

template <typename T>
void function();

// NO GOOD, can't do this!
function<test>();
```

Best Practices with Autogen

- Namespaces can be extended multiple times

```
namespace test {
  enum inner {
  };
}
```

Best Practices with Autogen

- Namespaces can be extended multiple times

```
namespace test {
  enum inner {
  };
}

namespace test {

  void func(inner a_EnumValue);
}
```

Best Practices with Autogen

- Namespaces can be extended multiple times

```
namespace test {
  enum inner {
  };
}

namespace test {
  // This works!
  void func(inner a_EnumValue);
}
```

Best Practices with Autogen

- Namespaces can be extended multiple times
- Structures require a single declaration

```
namespace test {
  enum inner {
  };
}

namespace test {
  // This works!
  void func(inner a_EnumValue);
}
```

```
struct test {
  enum inner {
  };
};
```

Best Practices with Autogen

- Namespaces can be extended multiple times
- Structures require a single declaration

```
namespace test {
  enum inner {
  };
}

namespace test {
  // This works!
  void func(inner a_EnumValue);
}
```

```
struct test {
  enum inner {
  };
};

struct test {

  void func(inner a_EnumValue);
};
```

Best Practices with Autogen

- Namespaces can be extended multiple times
- Structures require a single declaration

```
namespace test {
  enum inner {
  };
}

namespace test {
  // This works!
  void func(inner a_EnumValue);
}
```

```
struct test {
  enum inner {
  };
};

struct test {
  // Nope, struct is already declared
  void func(inner a_EnumValue);
};
```

Best Practices with Autogen

- Strongly typedef everything (userid, mechid, …)

Best Practices with Autogen

- Strongly typedef everything (userid, mechid, …)
  - Compile-time ‘apps hungarian’!

Best Practices with Autogen

- Autogenerate full, explicit constructors

Best Practices with Autogen

- Autogenerate full, explicit constructors
    - Especially for POD structures

Best Practices with Autogen

- Autogenerate full, explicit constructors
  - Especially for POD structures
  - Catches adding/removing data members

Best Practices with Autogen

- Autogenerate full, explicit constructors
  - Especially for POD structures
  - Catches adding/removing data members
  - Catches type changes with explicit

Best Practices with Autogen

- #line and #error directives

Best Practices with Autogen

- #line and #error directives
  - #line <#> <file>, magical, *cross platform*!

Best Practices with Autogen

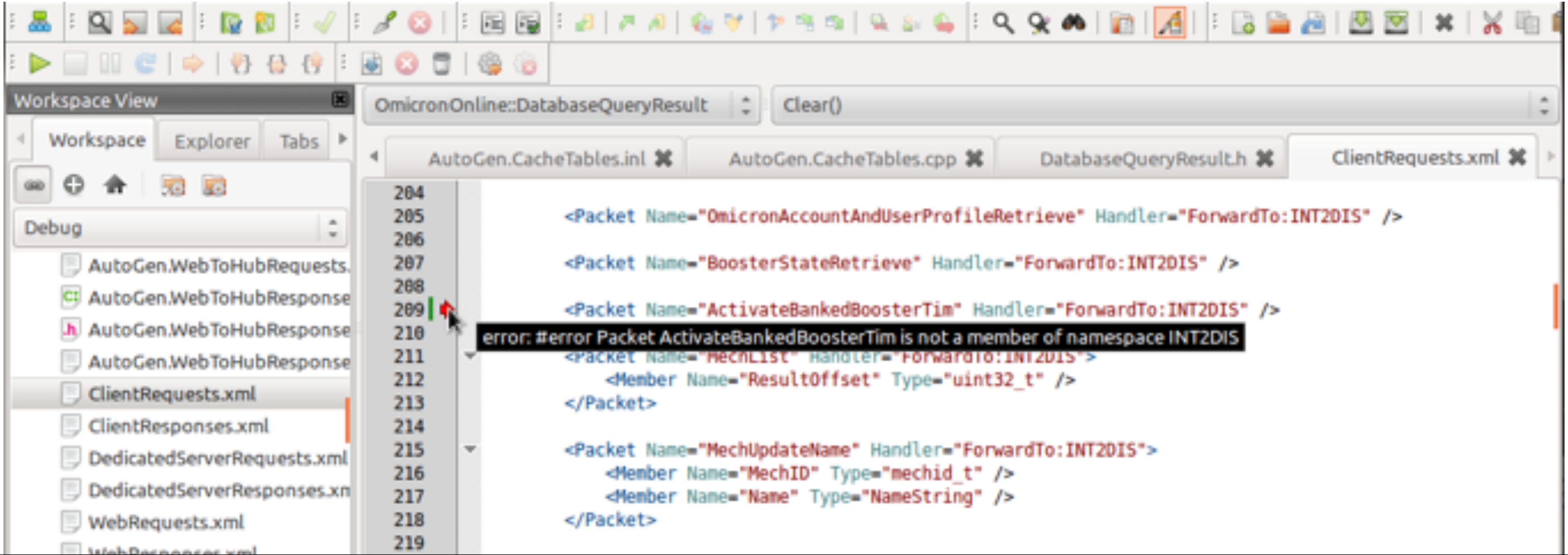
- #line and #error directives
  - #line <#> <file>, magical, *cross platform*!
  - #error <msg> to throw compiler error

Best Practices with Autogen

- #line and #error directives
  - #line <#> <file>, magical, *cross platform*!
  - #error <msg> to throw compiler error
  - Reference your data files

Best Practices with Autogen

- #line and #error directives



```
        <Packet Name="OmicronAccountAndUserProfileRetrieve" Handler="ForwardTo:INT2DIS" />

        <Packet Name="BoosterStateRetrieve" Handler="ForwardTo:INT2DIS" />

        <Packet Name="ActivateBankedBoosterTim" Handler="ForwardTo:INT2DIS" />

        <Packet Name="MechList" Handler="ForwardTo:INT2DIS">
            <Member Name="ResultOffset" Type="uint32_t" />
        </Packet>

        <Packet Name="MechUpdateName" Handler="ForwardTo:INT2DIS">
            <Member Name="MechID" Type="mechid_t" />
            <Member Name="Name" Type="NameString" />
        </Packet>
```

error: #error Packet ActivateBankedBoosterTim is not a member of namespace INT2DIS

Defining Packets with Autogen

- What turns a structure into a packet?

Defining Packets with Autogen

- What turns a structure into a packet?
  - For MWO, it requires a serialize method

Defining Packets with Autogen

- What turns a structure into a packet?
  - For MWO, it requires a serialize method
- What info is required?

Defining Packets with Autogen

- What turns a structure into a packet?
  - For MWO, it requires a serialize method
- What info is required?
  - A packet name

Defining Packets with Autogen

- What turns a structure into a packet?
  - For MWO, it requires a serialize method
- What info is required?
  - A packet name
  - A set of members

Defining Packets with Autogen

- What turns a structure into a packet?
  - For MWO, it requires a serialize method
- What info is required?
  - A packet name
  - A set of members
  - Members should have types

Defining Packets with Autogen

- Defining your templates

Defining Packets with Autogen

- Defining your templates
  - Want declaration, definition templates for C++

Defining Packets with Autogen

- Defining your templates
    - Want declaration, definition templates for C++
    - Potentially an inline template for speed

Defining Packets with Autogen

- Defining your templates
  - Want declaration, definition templates for C++
  - Potentially an inline template for speed
  - Remember to keep header size small!

Defining Packets with Autogen

```
<Packet Name=“Login">
   <Member Name=“Username" Type=“UsernameString" />
   <Member Name=“Password" Type=“PasswordString" />
</Packet>
```

Defining Packets with Autogen

```
<Packet Name=“Login">
   <Member Name=“Username" Type=“UsernameString" />
   <Member Name=“Password" Type=“PasswordString" />
</Packet>
```

```
foreach ($root->Packet as packet)
{



}
```

Defining Packets with Autogen

```
<Packet Name=“Login">
   <Member Name=“Username" Type=“UsernameString" />
   <Member Name=“Password" Type=“PasswordString" />
</Packet>
```

```
foreach ($root->Packet as packet)
{
  print(“bool “ . $packet.Name . ”::Serialize(ISerializer &a_Ser) {”);
  print(“  return“);




  print(“  true;“);
  print(“}“);
}
```

Defining Packets with Autogen

```
<Packet Name=“Login">
   <Member Name=“Username" Type=“UsernameString" />
   <Member Name=“Password" Type=“PasswordString" />
</Packet>
```

```
foreach ($root->Packet as packet)
{
  print(“bool “ . $packet.Name . ”::Serialize(ISerializer &a_Ser) {”);
  print(“  return“);
  foreach ($packet->Member as member)
  {
    print(“  a_Ser.Serialize(“ . $member.Name . “) && ”);
  }
  print(“  true;“);
  print(“}“);
}
```

Defining Packets with Autogen

```
<Packet Name=“Login">
   <Member Name=“Username" Type=“UsernameString" />
   <Member Name=“Password" Type=“PasswordString" />
</Packet>
```

```
{% for packet in root.iterchildren('Packet') %}
bool {{packet.attrib["Name"]}}::Serialize(ISerializer &a_Ser)
{
  return
{% for member in packet.iterchildren('Member') %}
    a_Ser.Serialize({{member.attrib["Name"]}}) &&
{% endfor %}
    true;
}
{% endfor %}
```

Dealing with a Stateless Design

- Adding metadata to packets

Dealing with a Stateless Design

- Adding metadata to packets
  - Method for embedding extra data in requests

Dealing with a Stateless Design

- Adding metadata to packets
  - Method for embedding extra data in requests
  - Called 'PacketSessionData' in MWO

Dealing with a Stateless Design

- Adding metadata to packets
  - Method for embedding extra data in requests
  - Called ‘PacketSessionData’ in MWO
  - Simply insert a container in packet header

Dealing with a Stateless Design

- Adding metadata to packets
  - May require rudimentary reflection

Dealing with a Stateless Design

- Adding metadata to packets
    - May require rudimentary reflection
    - Handlers should echo this data back

Dealing with a Stateless Design

- Adding metadata to packets
  - May require rudimentary reflection
  - Handlers should echo this data back
  - Keep it small!

Dealing with a Stateless Design

- Adding metadata to packets
    - May require rudimentary reflection
    - Handlers should echo this data back
    - Keep it small!
    - Clean up after yourself

Dealing with a Stateless Design

- Give in and add local state

Dealing with a Stateless Design

- Give in and add local state
  - For when metadata is just not enough

Dealing with a Stateless Design

- Give in and add local state
  - For when metadata is just not enough

Client:

```
<Packet Name=“RetrieveFriendsList">
 <Request>
  <Member Name=“UIDs" Type=“UIDList"/>
 </Request>
 <Response>
  <Member Name=“Names" Type=“UNameList"/>
 </Response>
</Packet>
```

Dealing with a Stateless Design

- Give in and add local state
  - For when metadata is just not enough

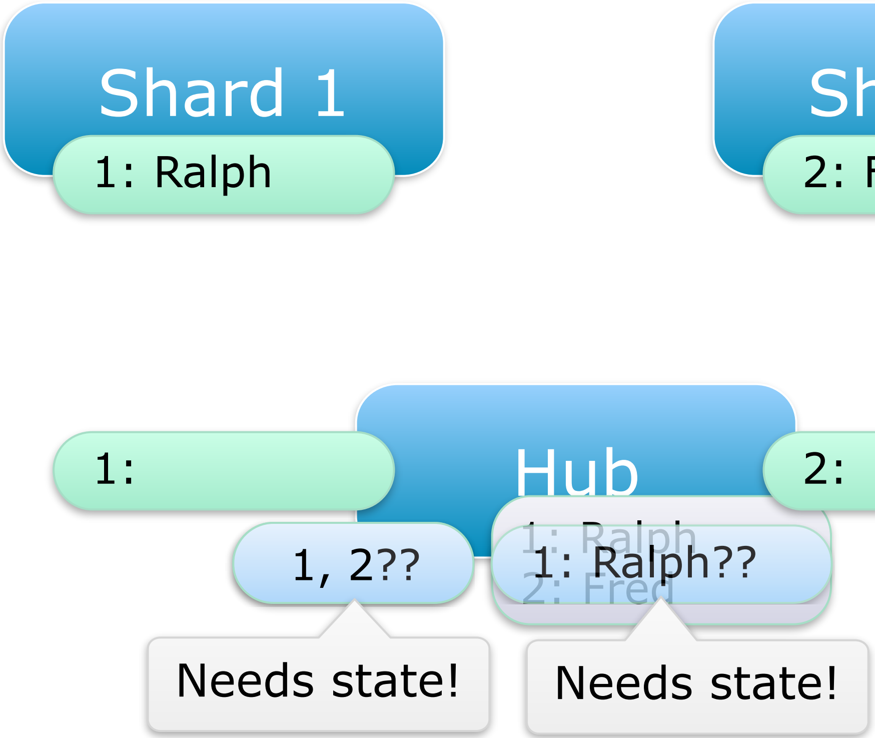
Client:

```
<Packet Name=“RetrieveFriendsList">
 <Request>
  <Member Name=“UIDs" Type=“UIDList"/>
 </Request>
 <Response>
  <Member Name=“Names" Type=“UNameList"/>
 </Response>
</Packet>
```

Persistent Storage:

```
<Packet Name=“RetrieveUserName">
 <Request>
  <Member Name=“UID" Type=“userid_t"/>
 </Request>
 <Response>
  <Member Name=“Name" Type=“UserName"/>
 </Response>
</Packet>
```

Dealing with a Stateless Design



Shard 1

1: Ralph

Shard 2

2: Fred

Hub

1:

2:

1, 2??

1: Ralph
2: Fred

1: Ralph??

Needs state!

Needs state!

Dealing with a Stateless Design

- Give in and add local state
  - For when metadata is just not enough
  - Keep a map or hash on server

Dealing with a Stateless Design

- Give in and add local state
  - For when metadata is just not enough
  - Keep a map or hash on server
  - Simple incrementing int to generate keys

Dealing with a Stateless Design

- Give in and add local state
  - For when metadata is just not enough
  - Keep a map or hash on server
  - Simple incrementing int to generate keys
  - Store key in packet metadata

Dealing with a Stateless Design

- Give in and add local state
    - Can’t always guarantee a response

Dealing with a Stateless Design

- Give in and add local state
  - Can’t always guarantee a response
  - Add a timeout mechanism

Dealing with a Stateless Design

- Give in and add local state
  - Can’t always guarantee a response
  - Add a timeout mechanism
  - Priority queue, sorted by timeout time

Dealing with a Stateless Design

- Give in and add local state
  - Can’t always guarantee a response
  - Add a timeout mechanism
  - Priority queue, sorted by timeout time
  - Pop from head until no longer timed out

Dealing with Asynchronous Code

- Problems with asynchronous design

Dealing with Asynchronous Code

- Problems with asynchronous design
  - Need to communicate between servers

Dealing with Asynchronous Code

- Problems with asynchronous design
  - Need to communicate between servers
  - Not allowed to block

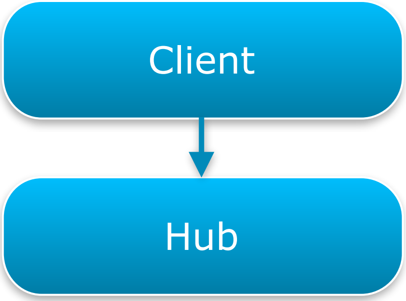
Dealing with Asynchronous Code

- Problems with asynchronous design
  - Need to communicate between servers
  - Not allowed to block
  - Serial logic broken around async points

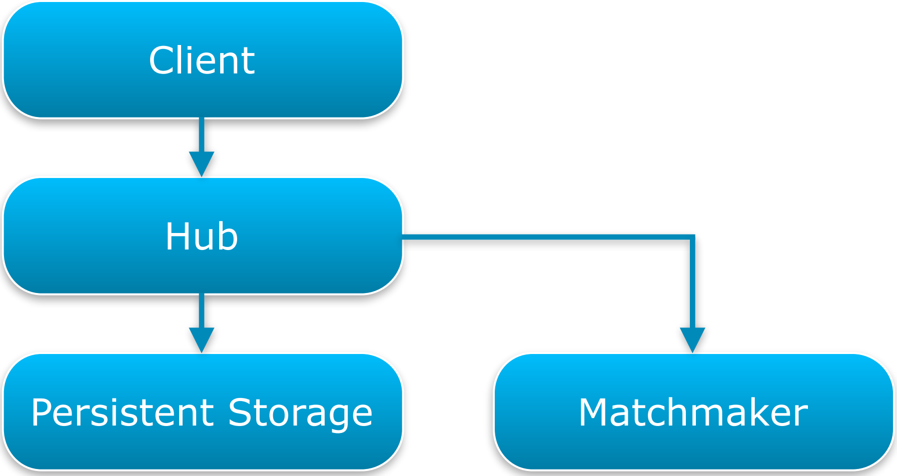
Dealing with Asynchronous Code

Client

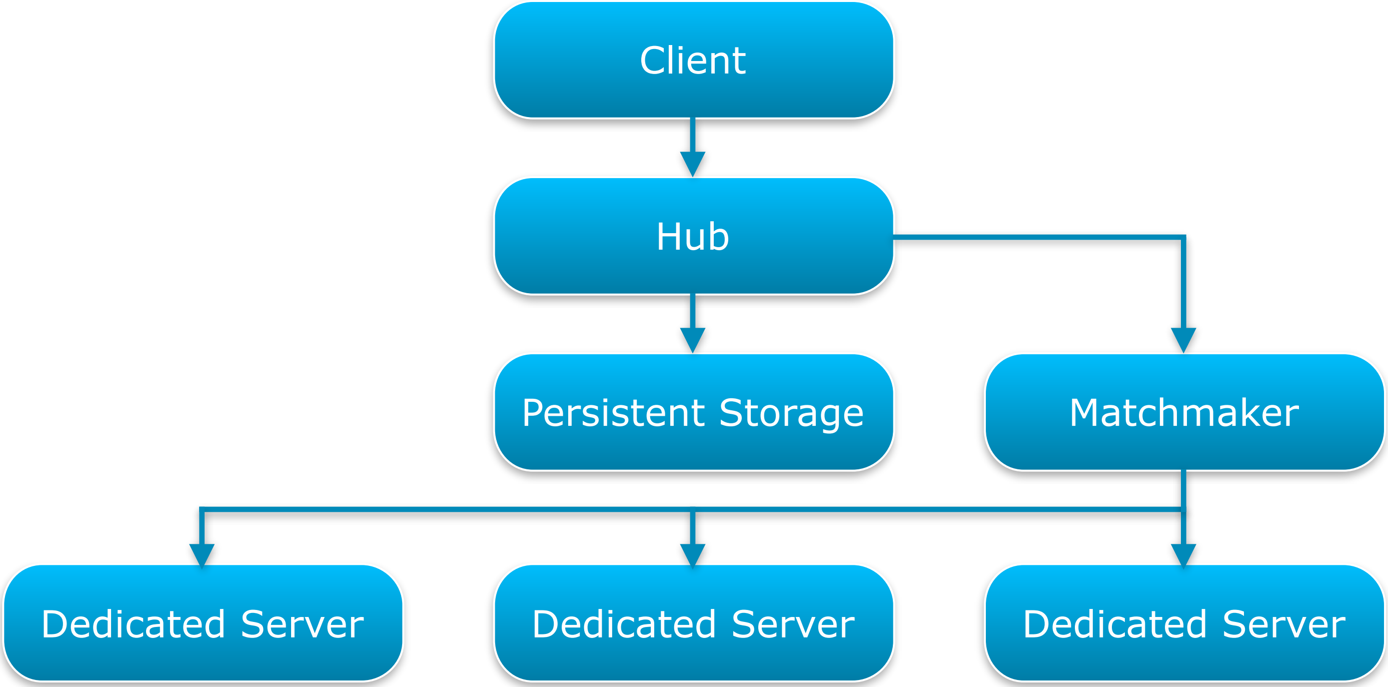
Dealing with Asynchronous Code



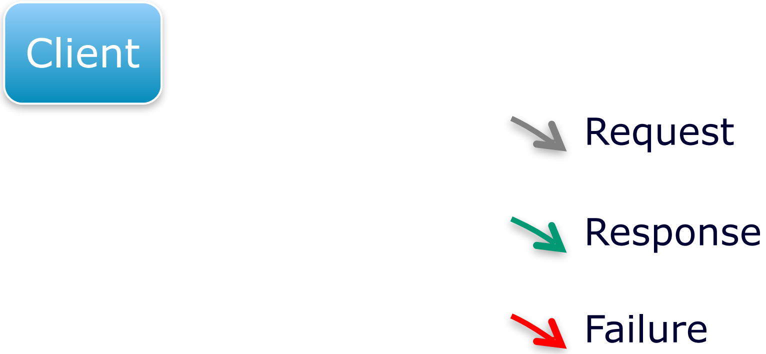
Dealing with Asynchronous Code



Dealing with Asynchronous Code



Dealing with Asynchronous Code



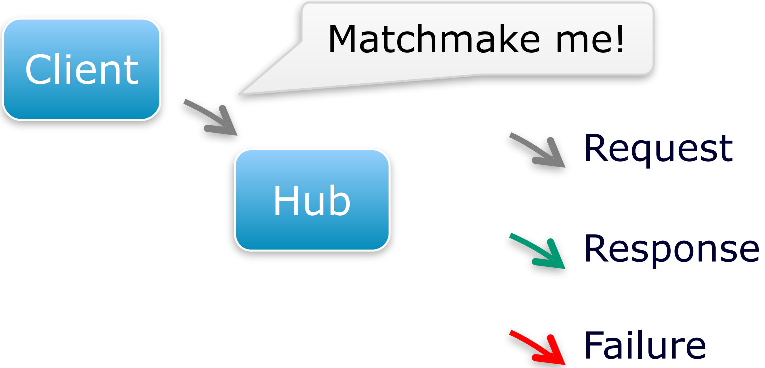
Client

Request

Response

Failure

Dealing with Asynchronous Code



Client

Matchmake me!

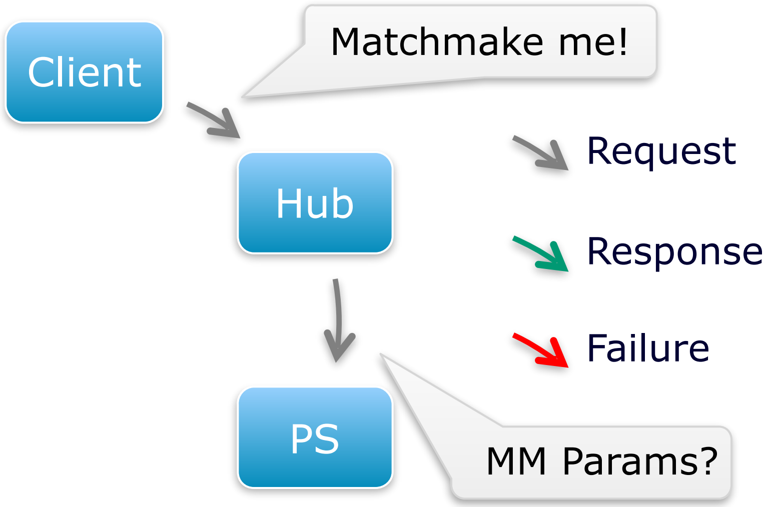
Hub

↘ Request

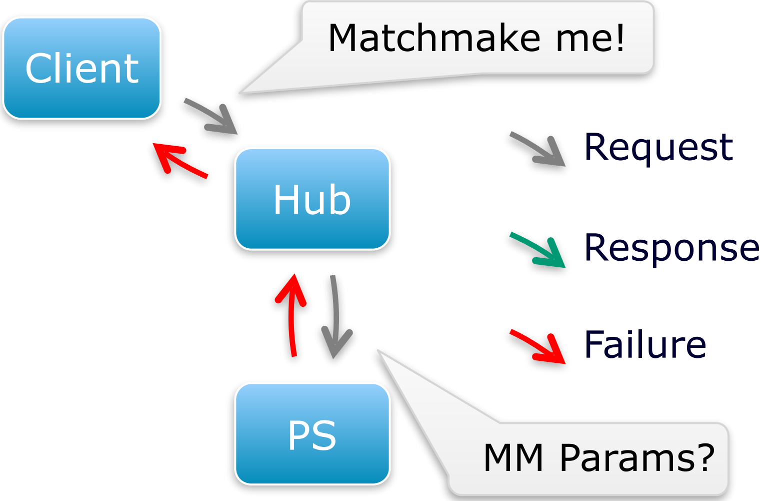
↘ Response

↘ Failure

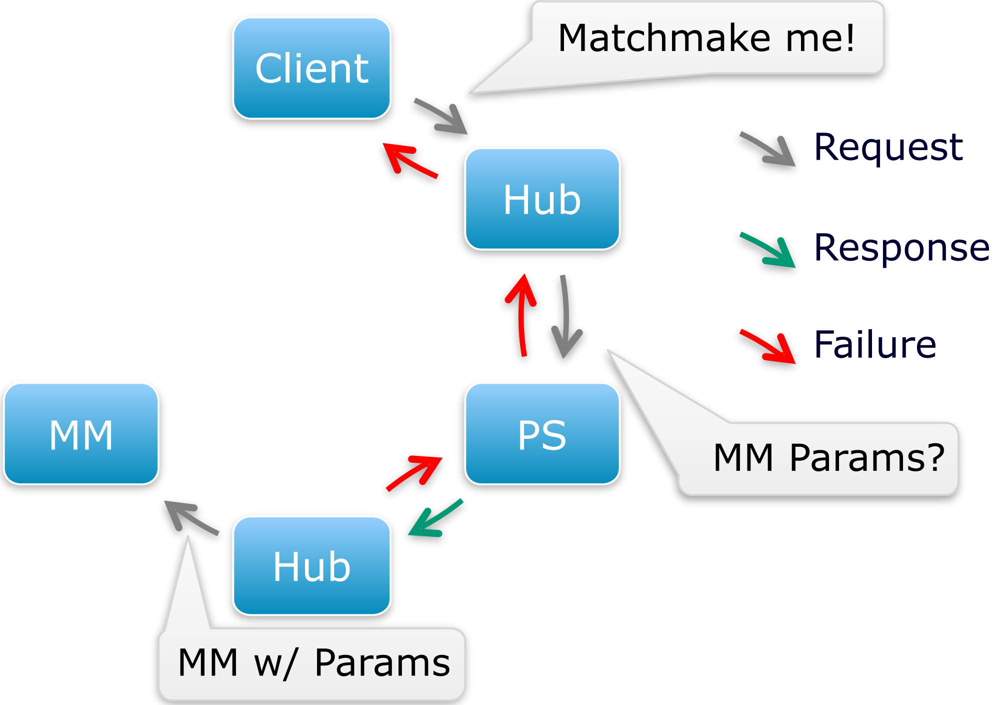
Dealing with Asynchronous Code



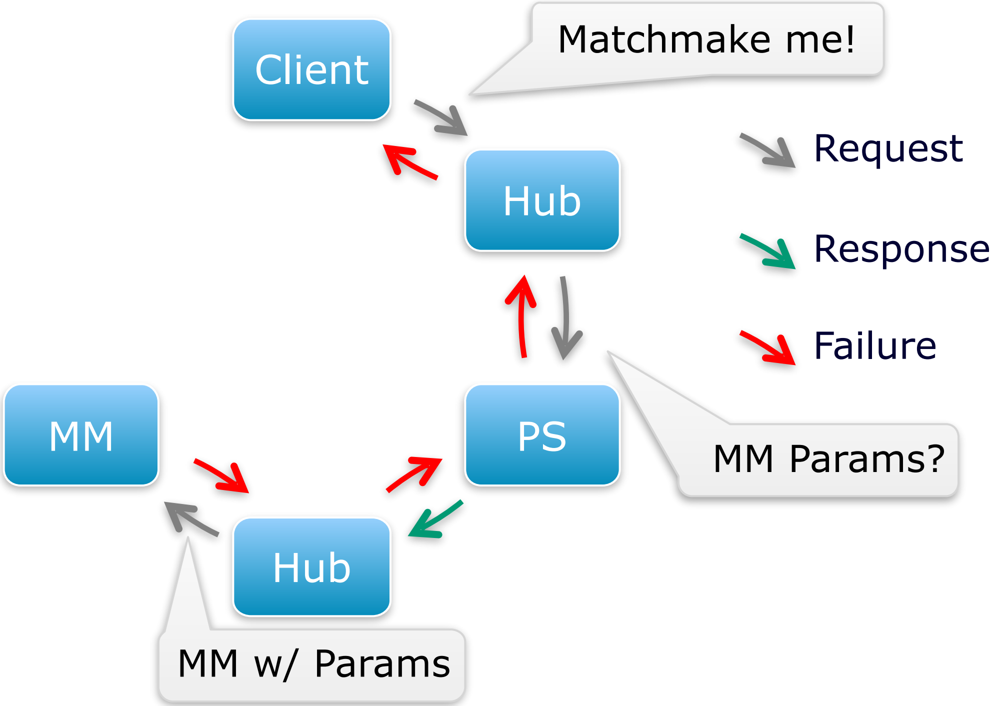
Dealing with Asynchronous Code



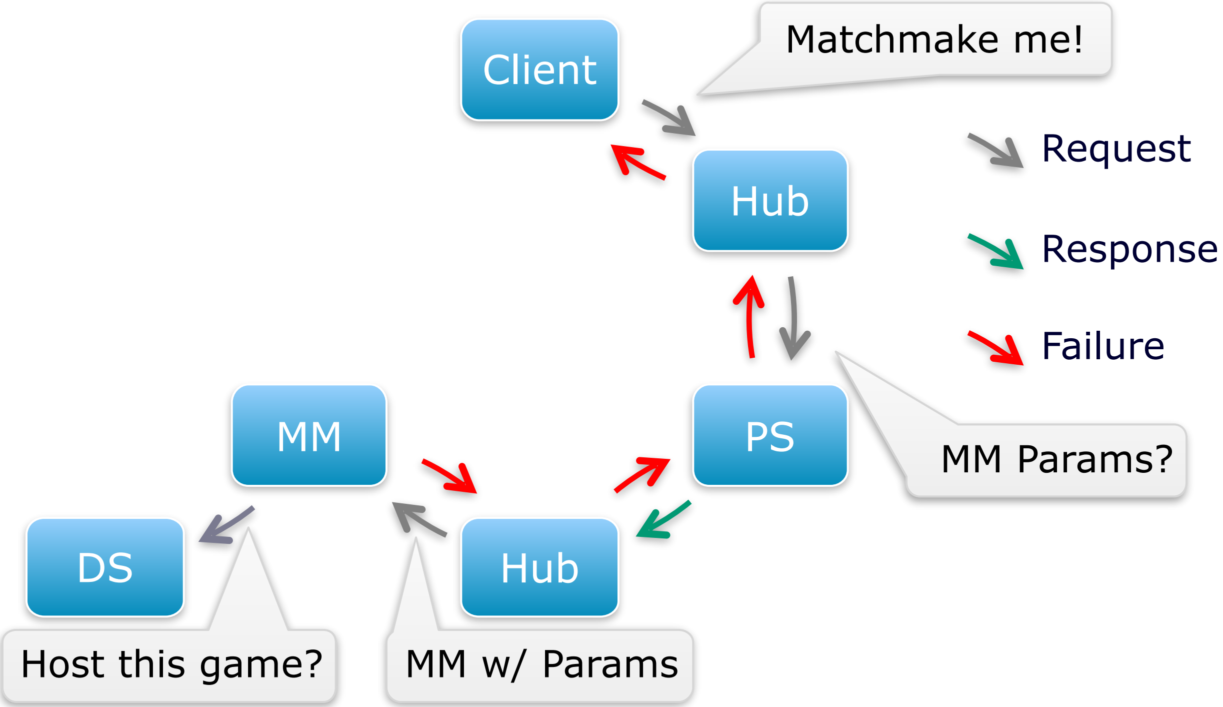
Dealing with Asynchronous Code



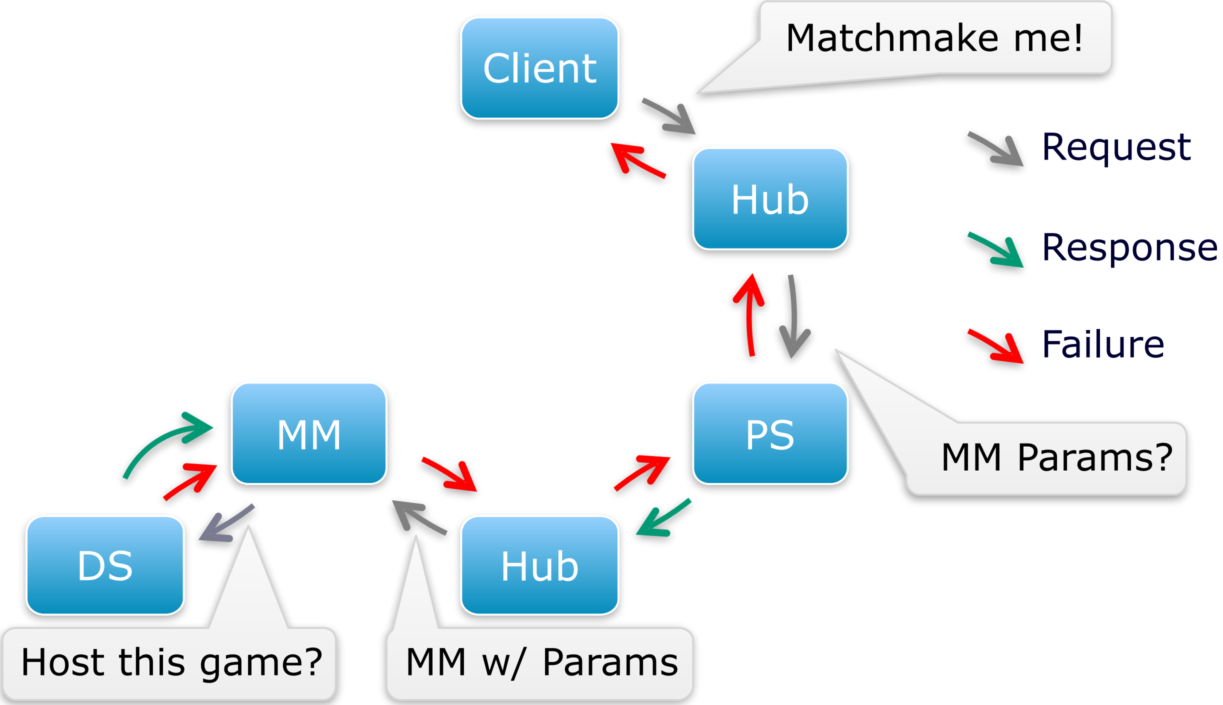
Dealing with Asynchronous Code



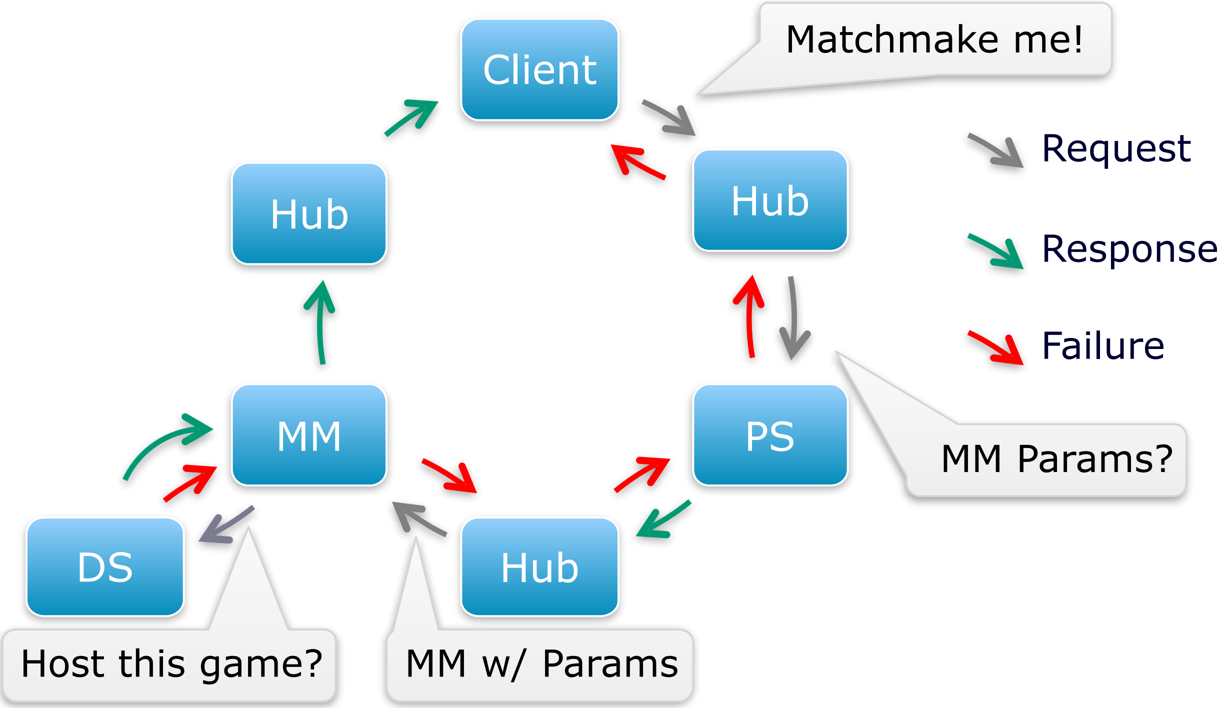
Dealing with Asynchronous Code



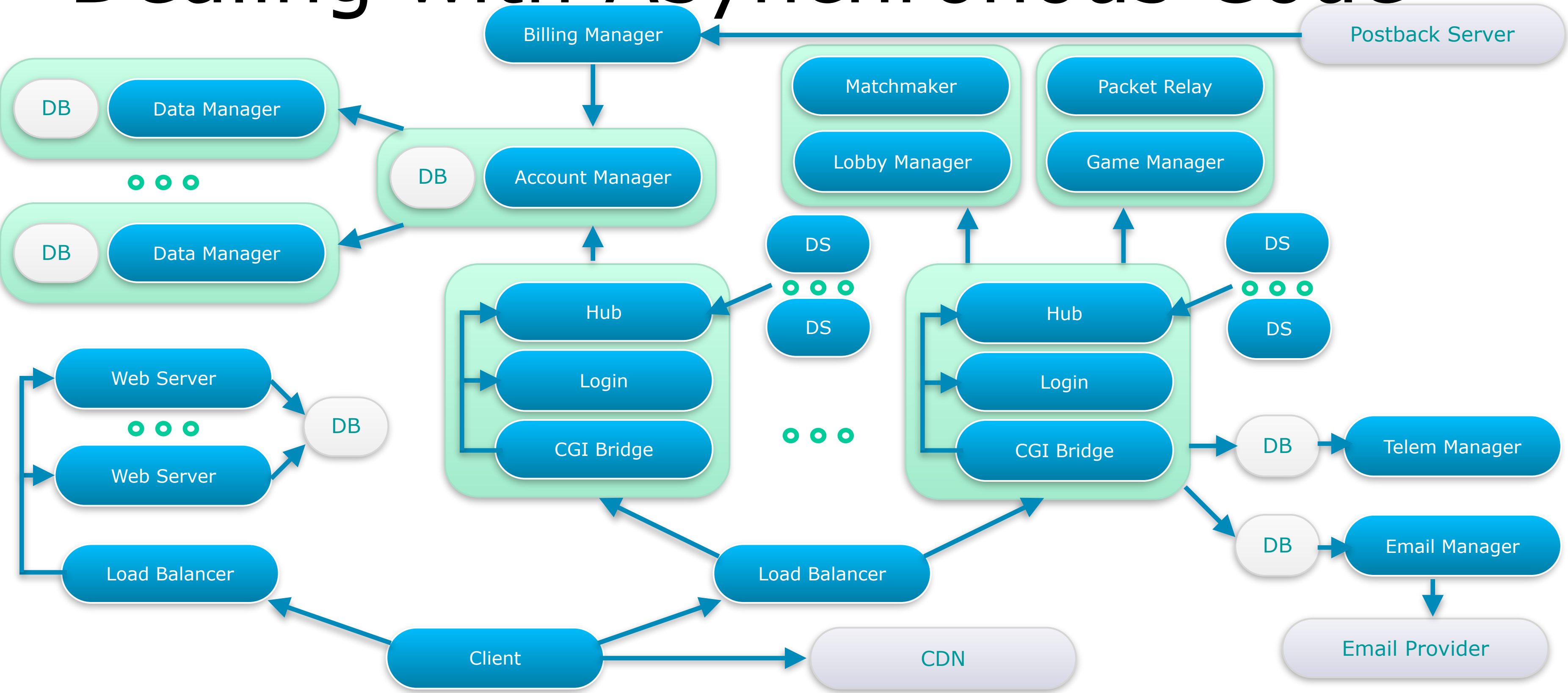
Dealing with Asynchronous Code



Dealing with Asynchronous Code



Dealing with Asynchronous Code



Dealing with Asynchronous Code

```
function Hub::HandleMatchmakeRequest(client, request)
{










}
```

Dealing with Asynchronous Code

```
function Hub::HandleMatchmakeRequest(client, request)
{
    mmParams = PS.Send( PS::MMParamsRetrieveRequest(request) );
    if (mmParams.failed) {
        return client.Send( Client::MMError(request, mmParams.errormsg) );
    }









}
```

Dealing with Asynchronous Code

```
function Hub::HandleMatchmakeRequest(client, request)
{
    mmParams = PS.Send( PS::MMParamsRetrieveRequest(request) );
    if (mmParams.failed) {
        return client.Send( Client::MMError(request, mmParams.errormsg) );
    }

    mmResult = MM.Send( MM::MMRequest(request, mmParams) );
    if (mmResult.failed) {
        return client.Send( Client::MMError(request, mmResult.errormsg) );
    }


}
```

Dealing with Asynchronous Code

```
function Hub::HandleMatchmakeRequest(client, request)
{
    mmParams = PS.Send( PS::MMParamsRetrieveRequest(request) );
    if (mmParams.failed) {
        return client.Send( Client::MMError(request, mmParams.errormsg) );
    }

    mmResult = MM.Send( MM::MMRequest(request, mmParams) );
    if (mmResult.failed) {
        return client.Send( Client::MMError(request, mmResult.errormsg) );
    }

    return client.Send( Client::MMResponse(request, mmResult) );
}
```

Dealing with Asynchronous Code

```
function MM::MakeGame() {







}
```

Dealing with Asynchronous Code

```
function MM::MakeGame() {
  MM::PlayerGameList list;
  if (MM::CreateGame(list)) {









  }
}
```

Dealing with Asynchronous Code

```
function MM::MakeGame() {
  MM::PlayerGameList list;
  if (MM::CreateGame(list)) {
    MM::DedicatedServerList serverList = MM::GetAvailableServers();
    foreach (serverList as server) {
      dsResult = server.Send( DS::ReserveForGame(list) );




      }
    }



  }
}
```

Dealing with Asynchronous Code

```
function MM::MakeGame() {
  MM::PlayerGameList list;
  if (MM::CreateGame(list)) {
    MM::DedicatedServerList serverList = MM::GetAvailableServers();
    foreach (serverList as server) {
      dsResult = server.Send( DS::ReserveForGame(list) );
      if (dsResult.success) {
        foreach (list as player)
          player.Hub.Send( Hub::MMResult(player, dsResult) );
      }
    }


  }
}
```

Dealing with Asynchronous Code

```
function MM::MakeGame() {
  MM::PlayerGameList list;
  if (MM::CreateGame(list)) {
    MM::DedicatedServerList serverList = MM::GetAvailableServers();
    foreach (serverList as server) {
      dsResult = server.Send( DS::ReserveForGame(list) );
      if (dsResult.success) {
        foreach (list as player)
          player.Hub.Send( Hub::MMResult(player, dsResult) );
      }
    }
    foreach (list as player)
      player.Hub.Send( Hub::MMFailed(player, “Failed”) );
  }
}
```

Dealing with Asynchronous Code

Dealing with Asynchronous Code

- Spawn a thread for each request?

Dealing with Asynchronous Code

- Spawn a thread for each request?
  - Uses lots of stack memory
  - Performance degrades

Dealing with Asynchronous Code

- Spawn a thread for each request?
  - Uses lots of stack memory
  - Performance degrades
- Resumable function?

Dealing with Asynchronous Code

- Spawn a thread for each request?
    - Uses lots of stack memory
    - Performance degrades
- Resumable function?
    - Function re-entrant from multiple points

Dealing with Asynchronous Code

- Spawn a thread for each request?
  - Uses lots of stack memory
  - Performance degrades
- Resumable function?
  - Function re-entrant from multiple points
  - Called a coroutine

Dealing with Asynchronous Code

- Goals for a coroutine

Dealing with Asynchronous Code

- Goals for a coroutine
  - Simple

Dealing with Asynchronous Code

- Goals for a coroutine
  - Simple
  - Cross platform

Dealing with Asynchronous Code

- Goals for a coroutine
  - Simple
  - Cross platform
  - Easy to use and debug

Dealing with Asynchronous Code

- Goals for a coroutine
  - Simple
  - Cross platform
  - Easy to use and debug
  - Abstract away asynchronous behaviour

Dealing with Asynchronous Code

- Approaches to coroutines in C++

Dealing with Asynchronous Code

- Approaches to coroutines in C++
  - Boost coroutine

Dealing with Asynchronous Code

- Approaches to coroutines in C++
    - Boost coroutine
    - setcontext() / makecontext()

Dealing with Asynchronous Code

- Approaches to coroutines in C++
  - Boost coroutine
  - setcontext() / makecontext()
  - Class with jump table using goto

Dealing with Asynchronous Code

- Approaches to coroutines in C++
  - Boost coroutine
  - setcontext() / makecontext()
  - Class with jump table using goto
  - Class with switch case

Dealing with Asynchronous Code

- Coroutines using switch

Dealing with Asynchronous Code

- Coroutines using switch
    - Cases will skip over flow control (Duff’s Device)

Dealing with Asynchronous Code

- Coroutines using switch
  - Cases will skip over flow control (Duff's Device)

```
register n = (count + 7) / 8; switch(count % 8) {
  case 0: do {  *to = *from++;
  case 7:       *to = *from++;
  case 6:       *to = *from++;
  case 5:       *to = *from++;
  case 4:       *to = *from++;
  case 3:       *to = *from++;
  case 2:       *to = *from++;
  case 1:       *to = *from++;
} while(--n > 0); }
```

Dealing with Asynchronous Code

- Coroutines using switch
    - This is not a new approach

Dealing with Asynchronous Code

- Coroutines using switch
  - This is not a new approach
  - Excellent article online by Simon Tatham

    http://www.chiark.greenend.org.uk/~sgtatham/coroutines.html

Dealing with Asynchronous Code

- Coroutines using switch
  - This is not a new approach
  - Excellent article online by Simon Tatham
    http://www.chiark.greenend.org.uk/~sgtatham/coroutines.html
  - Our goal is a safe implementation

Implementing Coroutines

- Defining a language

Implementing Coroutines

- Defining a language
  - Can leverage our autogeneration system!

Implementing Coroutines

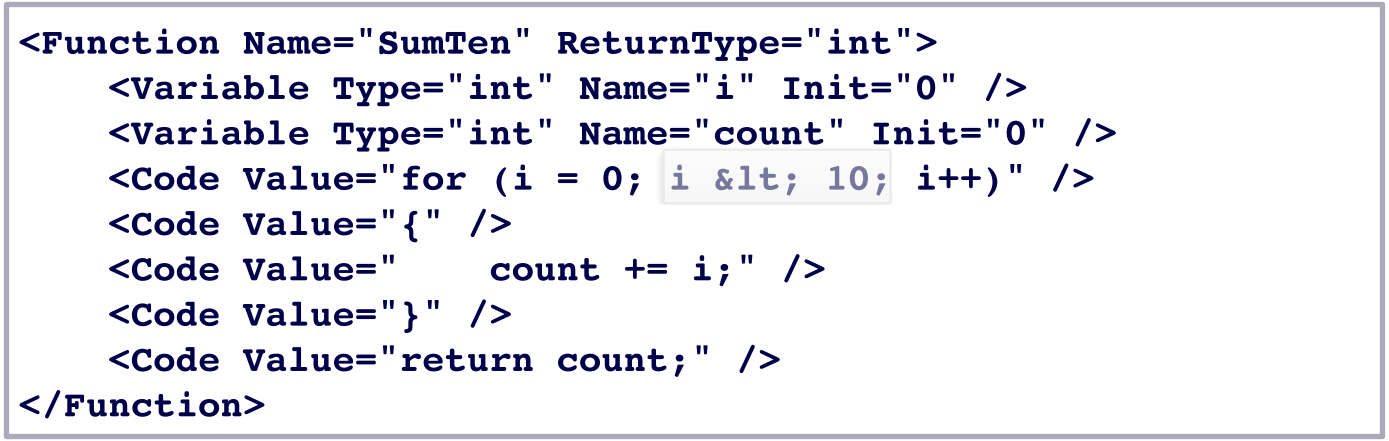
- Defining a language
  - Can leverage our autogeneration system!
  - But, data file can now contain flow control

Implementing Coroutines

- Defining a language
    - Can leverage our autogeneration system!
    - But, data file can now contain flow control
    - XML not necessarily the best fit

Implementing Coroutines

- Defining a language



```
<Function Name="SumTen" ReturnType="int">
    <Variable Type="int" Name="i" Init="0" />
    <Variable Type="int" Name="count" Init="0" />
    <Code Value="for (i = 0; i &lt; 10; i++)" />
    <Code Value="{" />
    <Code Value="    count += i;" />
    <Code Value="}" />
    <Code Value="return count;" />
</Function>
```

Implementing Coroutines

- Defining a language

```
<Function Name="SumTen" ReturnType="int">
    <Variable Type="int" Name="i" Init="0" />
    <Variable Type="int" Name="count" Init="0" />
    <For Init=“i = 0" Term=“i &lt; 10" Incr="i++" >
        <Sum Output="count" In1="count" In2="i" />
    </For>
    <Return Variable="count" />
</Function>
```

Implementing Coroutines

- Creating an instance

Implementing Coroutines

- Creating an instance
  - Make coroutine a timeout state structure

Implementing Coroutines

- Creating an instance
  - Make coroutine a timeout state structure
  - Store coroutines id in packet metadata

Implementing Coroutines

- Creating an instance
  - Make coroutine a timeout state structure
  - Store coroutines id in packet metadata
  - On response, fetch coroutine and resume! ?

Implementing Coroutines

- Creating an instance
    - Make coroutine a timeout state structure
    - Store coroutines id in packet metadata
    - On response, fetch coroutine and resume! ?
    - No, coroutine id’s will not be unique

Implementing Coroutines

- Identifying a coroutine owner

Implementing Coroutines

- Identifying a coroutine owner
    - Depends on your server architecture

Implementing Coroutines

- Identifying a coroutine owner
  - Depends on your server architecture
  - MWO 32-bit hash for any process

Implementing Coroutines

- Identifying a coroutine owner
  - Depends on your server architecture
  - MWO 32-bit hash for any process
  - Contains IPv4Address

Implementing Coroutines

- Identifying a coroutine owner
  - Depends on your server architecture
  - MWO 32-bit hash for any process
  - Contains IPv4Address
  - Service type

Implementing Coroutines

- Identifying a coroutine owner
  - Depends on your server architecture
  - MWO 32-bit hash for any process
  - Contains IPv4Address
  - Service type
  - Process ID

Implementing Coroutines

- Identifying a coroutine owner
  - Depends on your server architecture
  - MWO 32-bit hash for any process
  - Contains IPv4Address
  - Service type
  - Process ID
  - Store hash in metadata

Implementing Coroutines

- Handling timeouts

```
coroutine start
```

```
int i = 0;
FooResult results[2];
for (; i < 2; i++)
{
  FooRequest request;
  InvokeServer(request, results[i]);
}
```

Implementing Coroutines

- Handling timeouts

```
coroutine start
        - initialize loop
```

```
int i = 0;
FooResult results[2];
for (; i < 2; i++)
{
  FooRequest request;
  InvokeServer(request, results[i]);
}
```

Implementing Coroutines

- Handling timeouts

```
coroutine start
        - initialize loop
        - send request 1
```

```
int i = 0;
FooResult results[2];
for (; i < 2; i++)
{
  FooRequest request;
  InvokeServer(request, results[i]);
}
```

Implementing Coroutines

- Handling timeouts

```
coroutine start
        - initialize loop
        - send request 1
        - yield control
```

```
int i = 0;
FooResult results[2];
for (; i < 2; i++)
{
  FooRequest request;
  InvokeServer(request, results[i]);
}
```

Implementing Coroutines

- Handling timeouts

```
coroutine start
        - initialize loop
        - send request 1
        - yield control
        - timeout triggers
```

```
int i = 0;
FooResult results[2];
for (; i < 2; i++)
{
  FooRequest request;
  InvokeServer(request, results[i]);
}
```

Implementing Coroutines

- Handling timeouts

```
coroutine start
        – initialize loop
        – send request 1
        – yield control
        – timeout triggers
        – send request 2
```

```
int i = 0;
FooResult results[2];
for (; i < 2; i++)
{
  FooRequest request;
  InvokeServer(request, results[i]);
}
```

Implementing Coroutines

- Handling timeouts

```
coroutine start
        - initialize loop
        - send request 1
        - yield control
        - timeout triggers
        - send request 2
        - yield control
```

```
int i = 0;
FooResult results[2];
for (; i < 2; i++)
{
  FooRequest request;
  InvokeServer(request, results[i]);
}
```

Implementing Coroutines

- Handling timeouts

```
coroutine start
          – initialize loop
          – send request 1
          – yield control
          – timeout triggers
          – send request 2
          – yield control
          – receive request 1 response!
```

```
int i = 0;
FooResult results[2];
for (; i < 2; i++)
{
  FooRequest request;
  InvokeServer(request, results[i]);
}
```

Implementing Coroutines

- Handling timeouts
  - Need to uniquely identify *each request*

Implementing Coroutines

- Handling timeouts
  - Need to uniquely identify *each request*
  - Use a request counter

Implementing Coroutines

- Handling timeouts
    - Need to uniquely identify *each request*
    - Use a request counter
    - Can store counter in packet metadata

Implementing Coroutines

- Handling timeouts
  - Need to uniquely identify *each request*
  - Use a request counter
  - Can store counter in packet metadata
  - Only process response if counters match

Implementing Coroutines

- Handling timeouts
  - Need to uniquely identify *each request*
  - Use a request counter
  - Can store counter in packet metadata
  - Only process response if counters match
  - Increment on resume

Implementing Coroutines

```
coroutine start
```

Implementing Coroutines

```
coroutine start
         - coroutine.counter <- 0
```

Implementing Coroutines

```
coroutine start
        - coroutine.counter <- 0
        - initialize loop
```

Implementing Coroutines

```
coroutine start
        - coroutine.counter <- 0
        - initialize loop
        - send request 1
```

Implementing Coroutines

```
coroutine start
        - coroutine.counter <- 0
        - initialize loop
        - send request 1
                 - packet.counter <- coroutine.counter (0)
```

Implementing Coroutines

```
coroutine start
        - coroutine.counter <- 0
        - initialize loop
        - send request 1
                - packet.counter <- coroutine.counter (0)
        - yield control
```

Implementing Coroutines

```
coroutine start
        - coroutine.counter <- 0
        - initialize loop
        - send request 1
                - packet.counter <- coroutine.counter (0)
        - yield control
        - timeout triggers
```

Implementing Coroutines

```
coroutine start
        - coroutine.counter <- 0
        - initialize loop
        - send request 1
                 - packet.counter <- coroutine.counter (0)
        - yield control
        - timeout triggers
                 - coroutine.counter <- 1
```

Implementing Coroutines

```
coroutine start
        - coroutine.counter <- 0
        - initialize loop
        - send request 1
                - packet.counter <- coroutine.counter (0)
        - yield control
        - timeout triggers
                - coroutine.counter <- 1
        - send request 2
```

Implementing Coroutines

```
coroutine start
         - coroutine.counter <- 0
         - initialize loop
         - send request 1
                 - packet.counter <- coroutine.counter (0)
         - yield control
         - timeout triggers
                 - coroutine.counter <- 1
         - send request 2
                 - packet.counter <- coroutine.counter (1)
```

Implementing Coroutines

```
coroutine start
        - coroutine.counter <- 0
        - initialize loop
        - send request 1
                - packet.counter <- coroutine.counter (0)
        - yield control
        - timeout triggers
                - coroutine.counter <- 1
        - send request 2
                - packet.counter <- coroutine.counter (1)
        - yield control
```

Implementing Coroutines

```
coroutine start
        - coroutine.counter <- 0
        - initialize loop
        - send request 1
                - packet.counter <- coroutine.counter (0)
        - yield control
        - timeout triggers
                - coroutine.counter <- 1
        - send request 2
                - packet.counter <- coroutine.counter (1)
        - yield control
        - receive request 1 response!
```

Implementing Coroutines

```
coroutine start
        - coroutine.counter <- 0
        - initialize loop
        - send request 1
                - packet.counter <- coroutine.counter (0)
        - yield control
        - timeout triggers
                - coroutine.counter <- 1
        - send request 2
                - packet.counter <- coroutine.counter (1)
        - yield control
        - receive request 1 response!
                - packet.counter (0) != coroutine.counter (1)
```

Implementing Coroutines

```
coroutine start
        - coroutine.counter <- 0
        - initialize loop
        - send request 1
                - packet.counter <- coroutine.counter (0)
        - yield control
        - timeout triggers
                - coroutine.counter <- 1
        - send request 2
                - packet.counter <- coroutine.counter (1)
        - yield control
        - receive request 1 response!
                - packet.counter (0) != coroutine.counter (1), discard
```

Implementing Coroutines

```
coroutine start
        - coroutine.counter <- 0
        - initialize loop
        - send request 1
                - packet.counter <- coroutine.counter (0)
        - yield control
        - timeout triggers
                - coroutine.counter <- 1
        - send request 2
                - packet.counter <- coroutine.counter (1)
        - yield control
        - receive request 1 response!
                - packet.counter (0) != coroutine.counter (1), discard
        - receive request 2 response
```

Implementing Coroutines

```
coroutine start
        - coroutine.counter <- 0
        - initialize loop
        - send request 1
                - packet.counter <- coroutine.counter (0)
        - yield control
        - timeout triggers
                - coroutine.counter <- 1
        - send request 2
                - packet.counter <- coroutine.counter (1)
        - yield control
        - receive request 1 response!
                - packet.counter (0) != coroutine.counter (1), discard
        - receive request 2 response
                - packet.counter (1) == coroutine.counter (1)
```

Implementing Coroutines

```
coroutine start
        - coroutine.counter <- 0
        - initialize loop
        - send request 1
                - packet.counter <- coroutine.counter (0)
        - yield control
        - timeout triggers
                - coroutine.counter <- 1
        - send request 2
                - packet.counter <- coroutine.counter (1)
        - yield control
        - receive request 1 response!
                - packet.counter (0) != coroutine.counter (1), discard
        - receive request 2 response
                - packet.counter (1) == coroutine.counter (1), process
```

Example Code

http://static.mwomercs.com/img/karl/GDC2014.zip

Example Code

http://static.mwomercs.com/img/karl/GDC2014.zip

- Uses preprocessor!

Example Code

http://static.mwomercs.com/img/karl/GDC2014.zip

- Uses preprocessor!
  - Cross platform

Example Code

http://static.mwomercs.com/img/karl/GDC2014.zip

- Uses preprocessor!
    - Cross platform
    - No need to install a script runtime

Example Code

http://static.mwomercs.com/img/karl/GDC2014.zip

- Uses preprocessor!
  - Cross platform
  - No need to install a script runtime
  - Shows how simple autogeneration can be

Example Code

http://static.mwomercs.com/img/karl/GDC2014.zip

- Uses preprocessor!
  - Cross platform
  - No need to install a script runtime
  - Shows how simple autogeneration can be
  - It actually works!

Example Code

http://static.mwomercs.com/img/karl/GDC2014.zip

- Uses preprocessor!
  - Macro-based, ugly syntax

Example Code

http://static.mwomercs.com/img/karl/GDC2014.zip

- Uses preprocessor!
  - Macro-based, ugly syntax
  - Can’t handle hierarchy very well

Example Code

http://static.mwomercs.com/img/karl/GDC2014.zip

- Uses preprocessor!
    - Macro-based, ugly syntax
    - Can’t handle hierarchy very well
    - Weird token pasting rules

Example Code

http://static.mwomercs.com/img/karl/GDC2014.zip

- Uses preprocessor!
    - Macro-based, ugly syntax
    - Can’t handle hierarchy very well
    - Weird token pasting rules
    - Can’t emit comments, #line or #error

Example Code

http://static.mwomercs.com/img/karl/GDC2014.zip

- Uses preprocessor!
  - Macro-based, ugly syntax
  - Can’t handle hierarchy very well
  - Weird token pasting rules
  - Can’t emit comments, #line or #error
  - No support for conditionals in template

Future Work

Future Work

- Nicer coroutine syntax

Future Work

- Nicer coroutine syntax
- Issuing parallel requests from a coroutine

Future Work

- Nicer coroutine syntax
- Issuing parallel requests from a coroutine
- Could you write a coroutine serializer?

Future Work

- Nicer coroutine syntax
- Issuing parallel requests from a coroutine
- Could you write a coroutine serializer?
    - Why?

Questions!

karl.berg@piranhagames.com