Network Serialization and Routing in World of Warcraft

Joe Rumsey
jrumsey@blizzard.com
Twitter: @joerumz

What is JAM?

Joe's

Automated

Messages

The Problem

Game servers need to communicate with each other

Manual serialization is error-prone

```
void Serialize(stream &msg)
{
    vector<int> values;
    // ...Fill in some values...
    msg << values.size();
    for(int i = values.size(); --i;)
    {
        msg << values[i];
    }
}
```

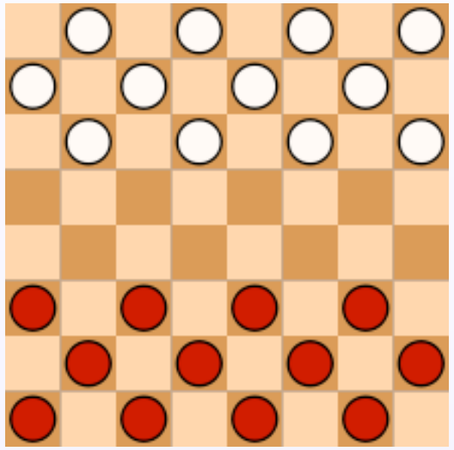
```
void Deserialize(stream &msg)
{
    vector<int> values;
    int size;
    msg >> size;
    values.resize(size);
    for(int i = size; i--;)
    {
        msg >> values[i];
    }
}
```

Manual serialization is error-prone

```
void Serialize(stream &msg)
{
    vector<int> values;
    // ...Fill in some values...
    msg << values.size();
    for(int i = values.size(); --i;)
    {
        msg << values[i];
    }
}
```

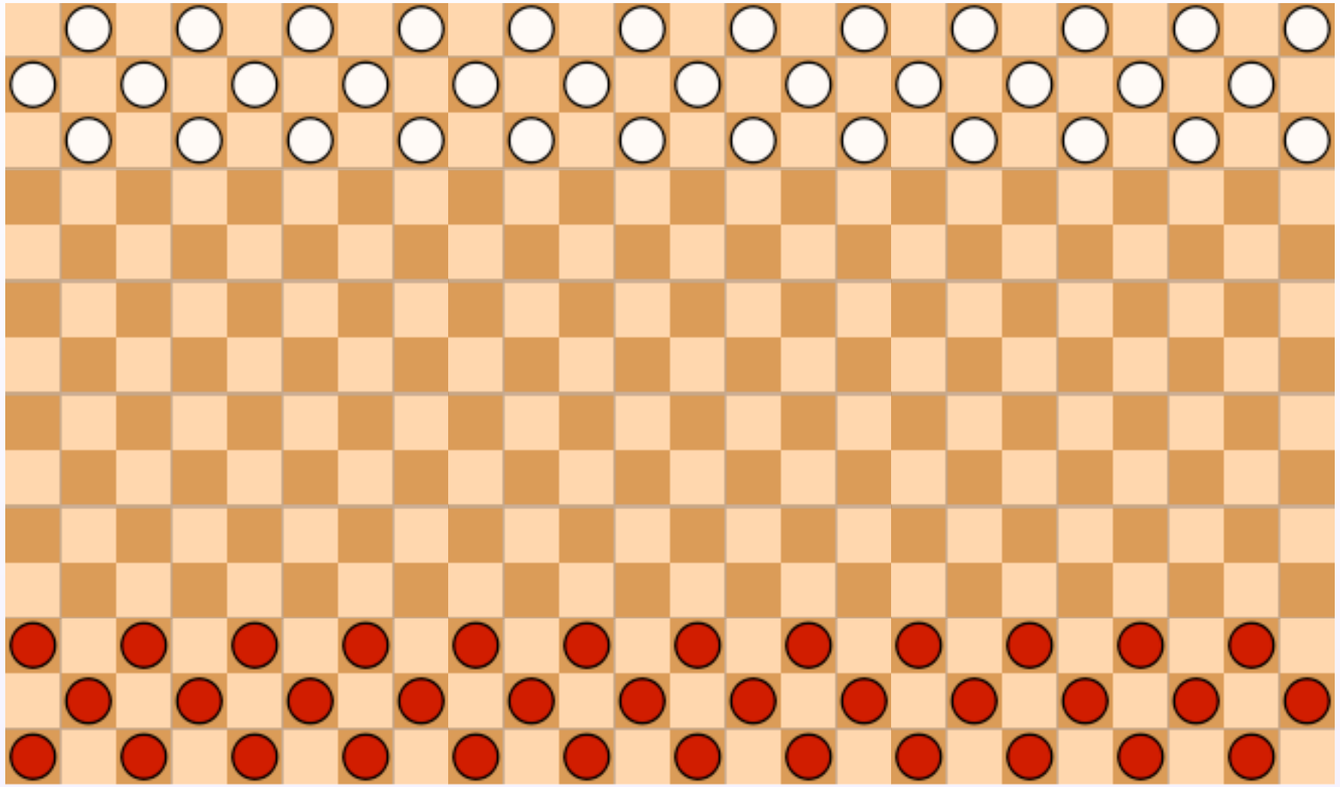
```
void Deserialize(stream &msg)
{
    vector<int> values;
    int size;
    msg >> size;
    values.resize(size);
    for(int i = size; i--;)
    {
        msg >> values[i];
    }
}
```

Manual serialization doesn't scale



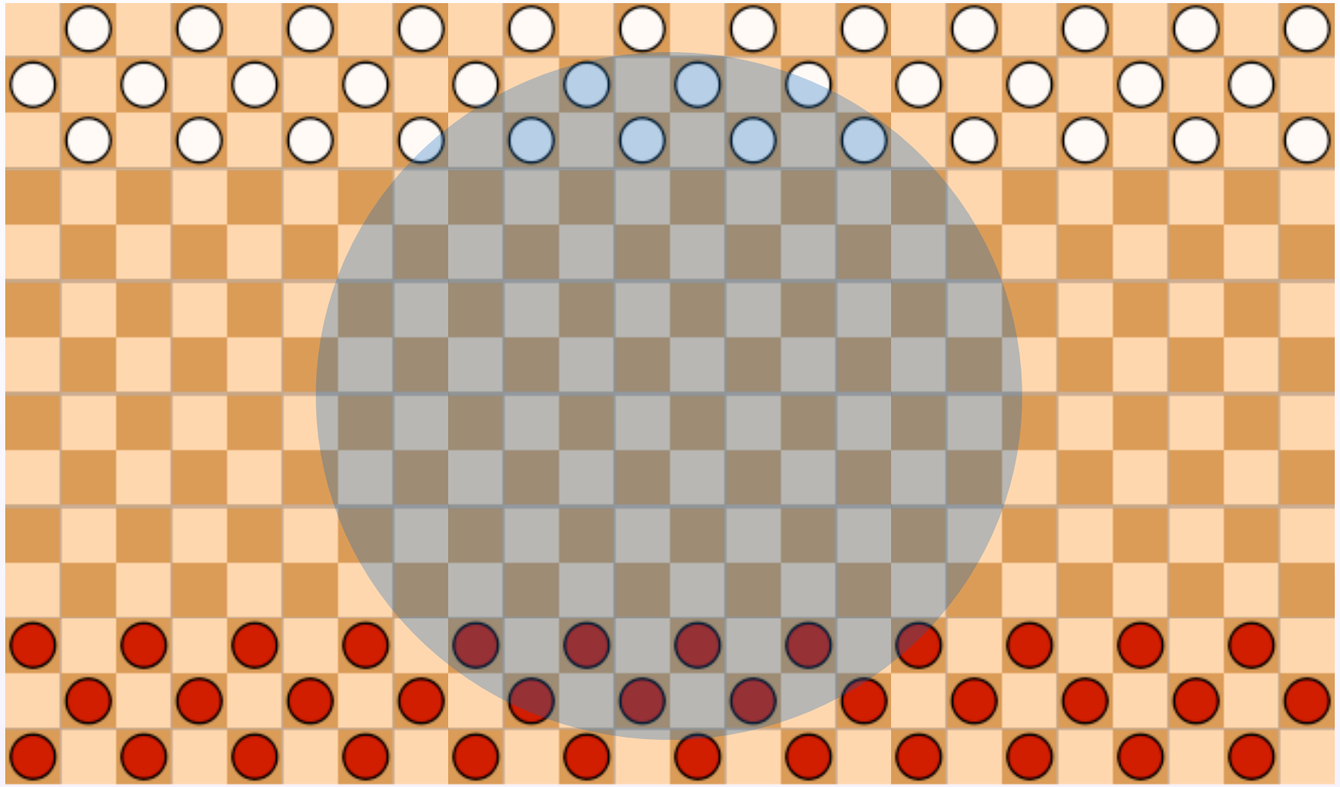
Manual serialization doesn't scale

World Of Checkers



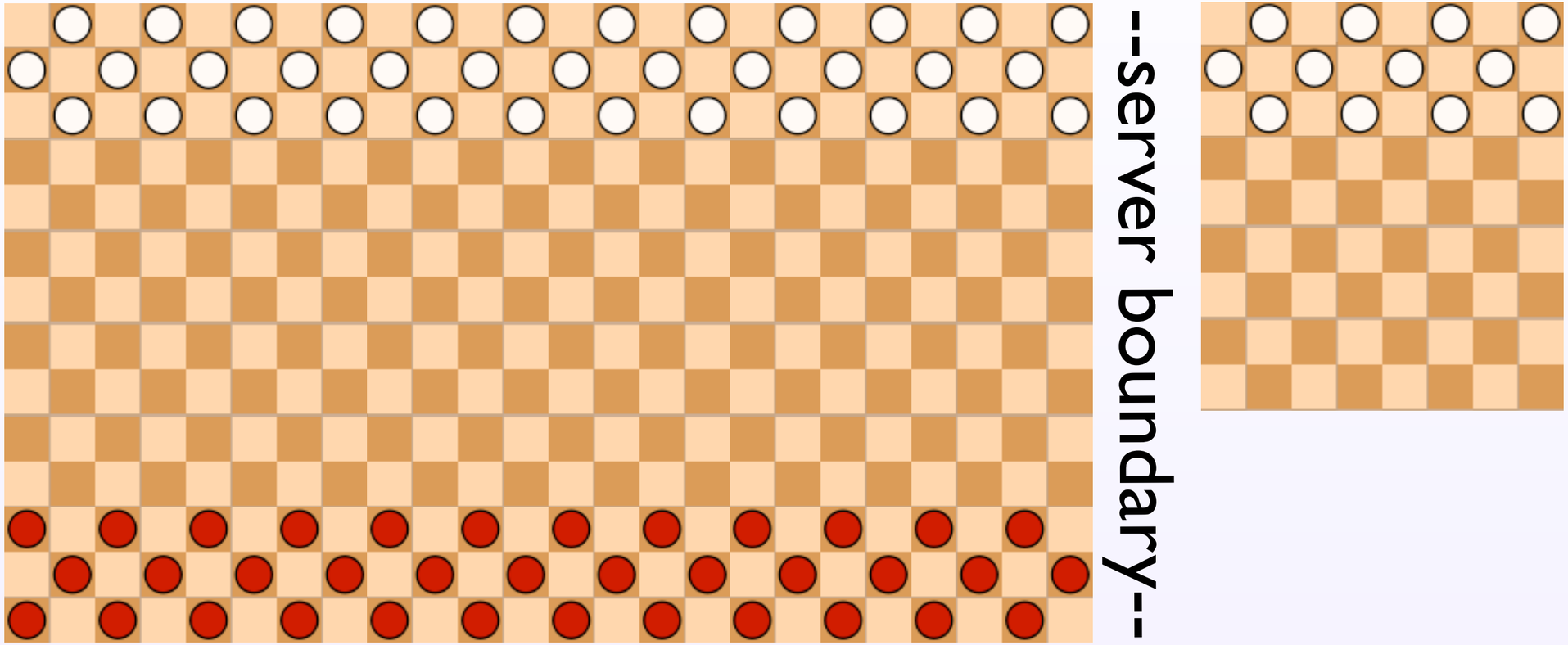
Manual serialization doesn't scale

World Of Checkers



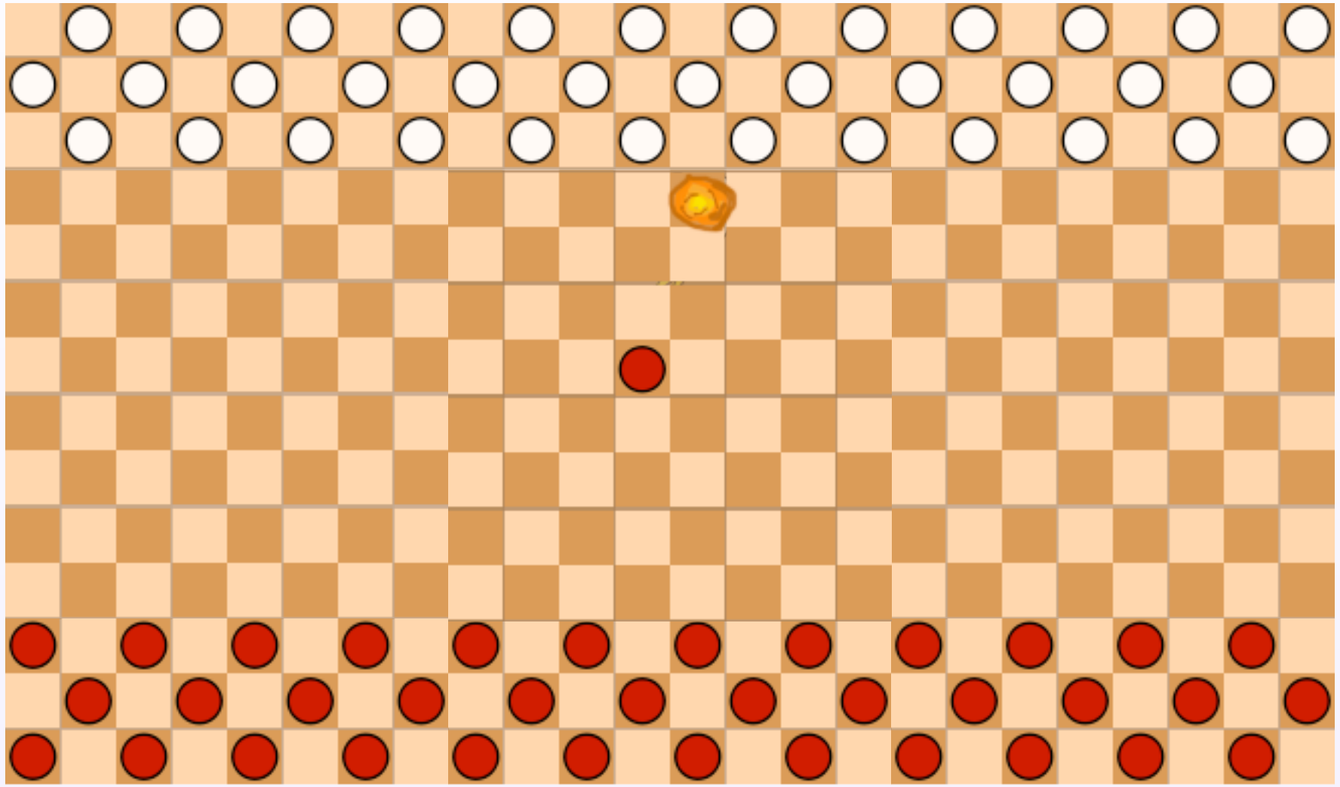
Manual serialization doesn't scale

World Of Checkers



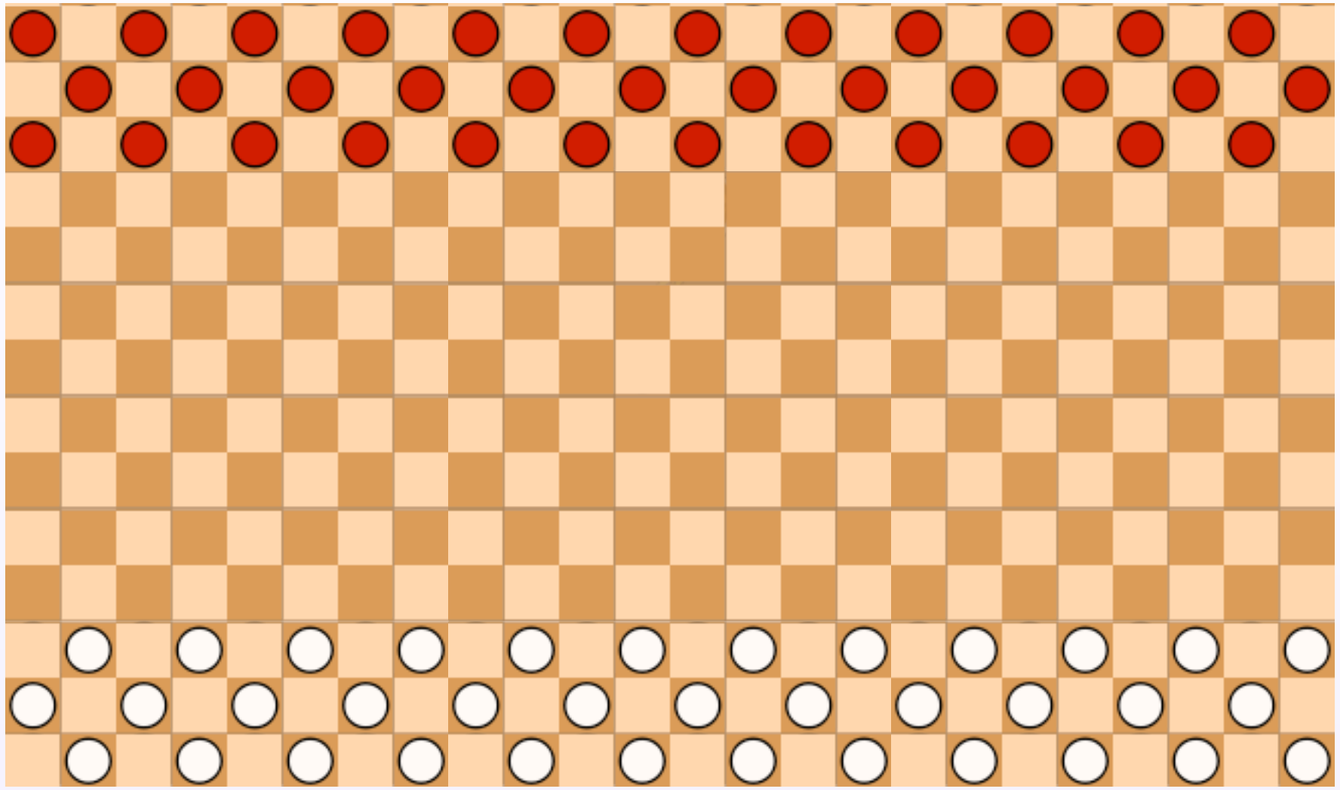
Manual serialization doesn't scale

World Of Checkers



Manual serialization doesn't scale

World Of Checkers



Goals

Goals

- DRY - Don't Repeat Yourself

Goals

- DRY - Don't Repeat Yourself
- Eliminate boilerplate to reduce bugs

Goals

- DRY - Don't Repeat Yourself
- Eliminate boilerplate to reduce bugs
- No more hand-coded serialize/deserialize

Goals

- DRY - Don't Repeat Yourself
- Eliminate boilerplate to reduce bugs
- No more hand-coded serialize/deserialize
- Spend more time on the game, not the protocol

Goals

- DRY - Don't Repeat Yourself
- Eliminate boilerplate to reduce bugs
- No more hand-coded serialize/deserialize
- Spend more time on the game, not the protocol
- Build a helpful robot that writes our code for us

Goal: Human readable code

```
struct CheckerCaptured {
    CheckerID id;
    CheckerID capturedBy;
    u8 jumpType;
};

void Capture(CheckerID id, CheckerID by, JUMP_TYPE jumpType)
{
    CheckerCaptured msg;
    msg.id = id;
    msg.capturedBy = by;
    msg.jumpType = jumpType;
    Send(&msg);
}
```

Goal: Human readable code

```
struct CheckerCaptured {
    CheckerID id;
    CheckerID capturedBy;
    u8 jumpType;
};
```

```
void Capture(CheckerID id, CheckerID by, JUMP_TYPE jumpType)
{
    CheckerCaptured msg;
    msg.id = id;
    msg.capturedBy = by;
    msg.jumpType = jumpType;
    Send(&msg);
}
```

Goal: Human readable code

```
struct CheckerCaptured {
    CheckerID id;
    CheckerID capturedBy;
    u8 jumpType;
};

void Capture(CheckerID id, CheckerID by, JUMP_TYPE jumpType)
{
    CheckerCaptured msg;
    msg.id = id;
    msg.capturedBy = by;
    msg.jumpType = jumpType;
    Send(&msg);
}
```

Goal: Human readable code

```
struct CheckerCaptured {
    CheckerID id;
    CheckerID capturedBy;
    u8 jumpType;
};

void Capture(CheckerID id, CheckerID by, JUMP_TYPE jumpType)
{
    CheckerCaptured msg;
    msg.id = id;
    msg.capturedBy = by;
    msg.jumpType = jumpType;
    Send(&msg);
}
```

Goal: Human readable code

```
struct CheckerCaptured {
    CheckerID id;
    CheckerID capturedBy;
    u8 jumpType;
};

void Capture(CheckerID id, CheckerID by, JUMP_TYPE jumpType)
{
    CheckerCaptured msg;
    msg.id = id;
    msg.capturedBy = by;
    msg.jumpType = jumpType;
    Send(&msg);
}
```

Goal: Human readable code

```
struct CheckerCaptured {
    CheckerID id;
    CheckerID capturedBy;
    u8 jumpType;
};

void Capture(CheckerID id, CheckerID by, JUMP_TYPE jumpType)
{
    CheckerCaptured msg;
    msg.id = id;
    msg.capturedBy = by;
    msg.jumpType = jumpType;
    Send(&msg);
}
```

Implementation Details

Development Cycle

- Describe the protocol
- Generate serialization and dispatch
- Send messages
- Receive messages
- Configure routing info

1-to-1 mapping of .jam messages to C++ classes

```
// From Checkers.jam
message CheckerCaptureCredit {
    CheckerID capturedCheckerID;
    CheckerID capturedBy;
    u8 jumpType;
};
```

1-to-1 mapping of .jam messages to C++ classes

```
// From Checkers.jam
message CheckerCaptureCredit {
    CheckerID capturedCheckerID;
    CheckerID capturedBy;
    u8 jumpType;
};
```

```
// 100% Generated code in JamCheckers.cpp
class CheckerCaptureCredit : public JamMessage {
public:
    // Message decoders
    BOOL Get(BinaryDecoder &decoder);
    BOOL Get(JSONDecoder &decoder);
    // Message encoders
    BOOL Put(BinaryEncoder &encoder) const;
    BOOL Put(JSONEncoder &encoder) const;
    /**** DATA START ****/
    CheckerID capturedCheckerID;
    CheckerID capturedBy;
    u8 jumpType;
    /**** DATA STOP  ****/
    // Lots more stuff...
};
```

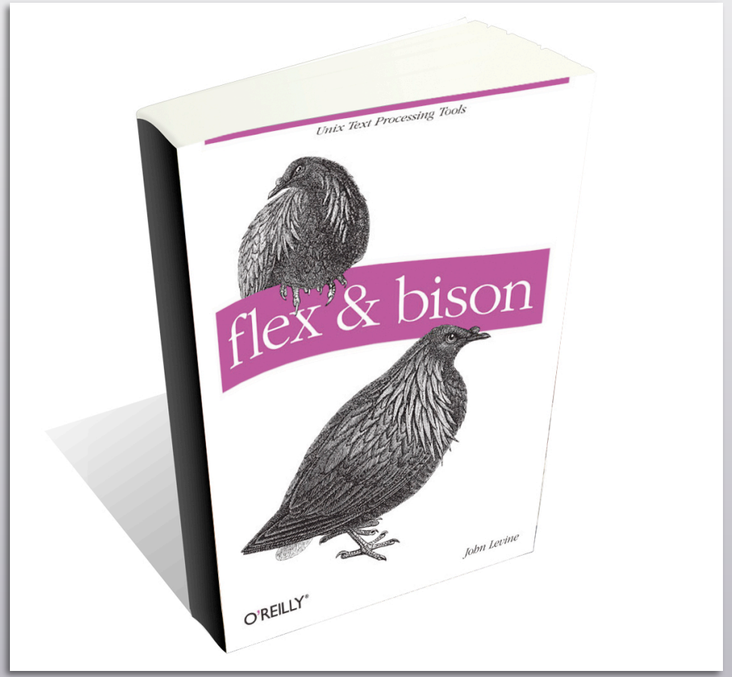
Development Cycle

- Describe the protocol
- Generate serialization and dispatch
- Send messages
- Receive messages
- Configure routing info

Auto-generated serialization code

```
//NOTICE: This is generated code.  DO NOT EDIT!
BOOL CheckerCaptureCredit::Put(BinaryEncoder &_encoder) const
{
    _encoder.BeginMessage(CODE, NAME);
    _encoder.Put("capturedCheckerID", capturedCheckerID);
    _encoder.Put("capturedBy", capturedBy);
    _encoder.Put("jumpType", jumpType);
    _encoder.EndMessage(CODE, NAME);
    return TRUE;
}
```

Flex and Bison make writing parsers easy



Flex & Bison - parser generators

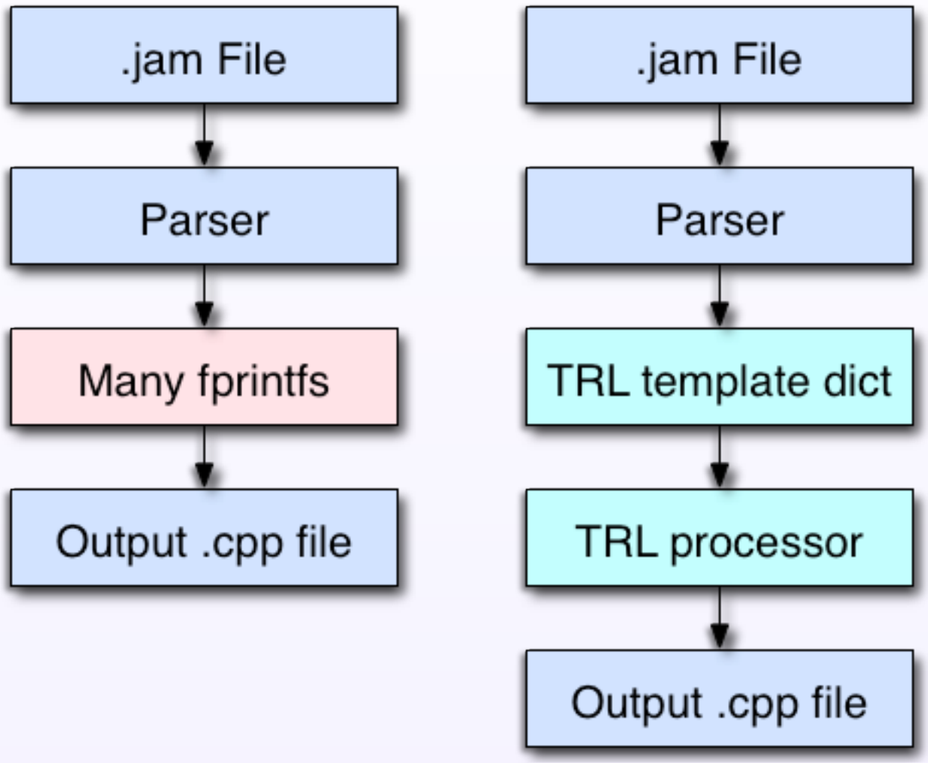
Other tools

- ANTLR
- GOLD
- PLY (Python Lex & Yacc)
- Boost.Spirit

JAM File syntax is described to Bison

```
jamstructs      : jamstructs jamstruct
                | jamstruct
                ;

jamstruct       : structtype TIDENTIFIER messagename '{' { jam_start_s
                | structtype TIDENTIFIER messagename '{' '}' ';' { jam
                | jamcomment { /*printf("Comment\n");*/ }
                ;

structtype      : TMESSAGE { $$.type = GMESSAGE; $$.qualifiers = 0; }
                | TOBJECTMESSAGE { $$.type = GMESSAGE; $$.qualifiers =
                | TSTRUCT { $$.type = GSTRUCT; $$.qualifiers = 0;}
                ;
```

From .jam to .cpp



2004 2013

TRL Turns .jam into C++

```
{@ define OutputMessage(msg, encoders, decoders) @}
    //
    // NOTICE: This is generated code. DO NOT EDIT!
    //
    class {{ msg.structName }} : public JamMessage {
    public:
        static u32 CRC;
        static u16 CODE;
        static cchar *NAME;

        // No argument constructor:
        {{ msg.structName }}() {

            {@ foreach f in msg.fields @}
                {@ if f.hasDefault @}
                    {{ f.name }} = {{ f.defValue }};
                {@ end if @}
            {@ end foreach @}

        }
```

TRL to generate a message class definition

TRL Turns .jam into C++

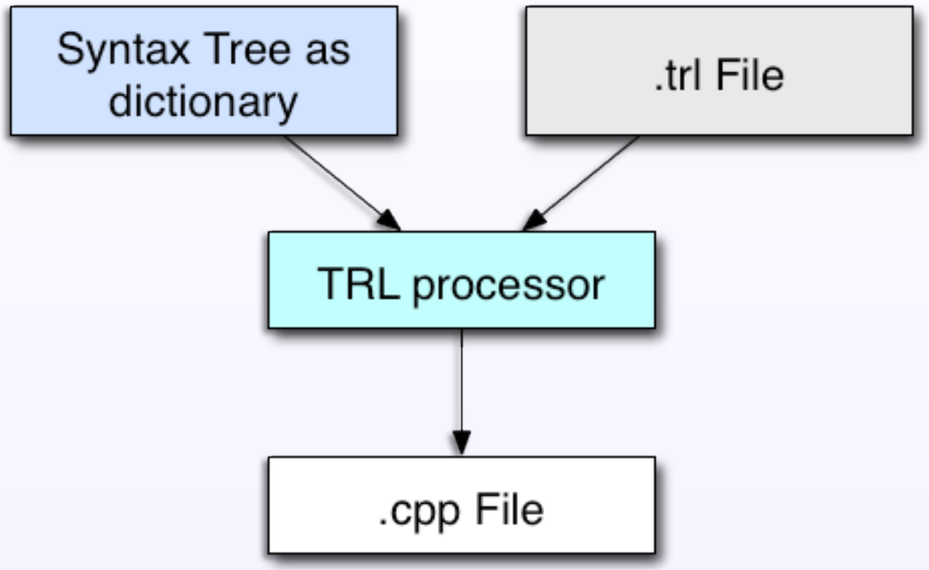
```
{@ define OutputMessage(msg, encoders, decoders) @}
    //
    // NOTICE: This is generated code. DO NOT EDIT!
    //
    class {{ msg.structName }} : public JamMessage {
    public:
        static u32 CRC;
        static u16 CODE;
        static cchar *NAME;

        // No argument constructor:
        {{ msg.structName }}() {

            {@ foreach f in msg.fields @}
                {@ if f.hasDefault @}
                    {{ f.name }} = {{ f.defValue }};
                {@ end if @}
            {@ end foreach @}

        }
```

TRL to generate a message class definition

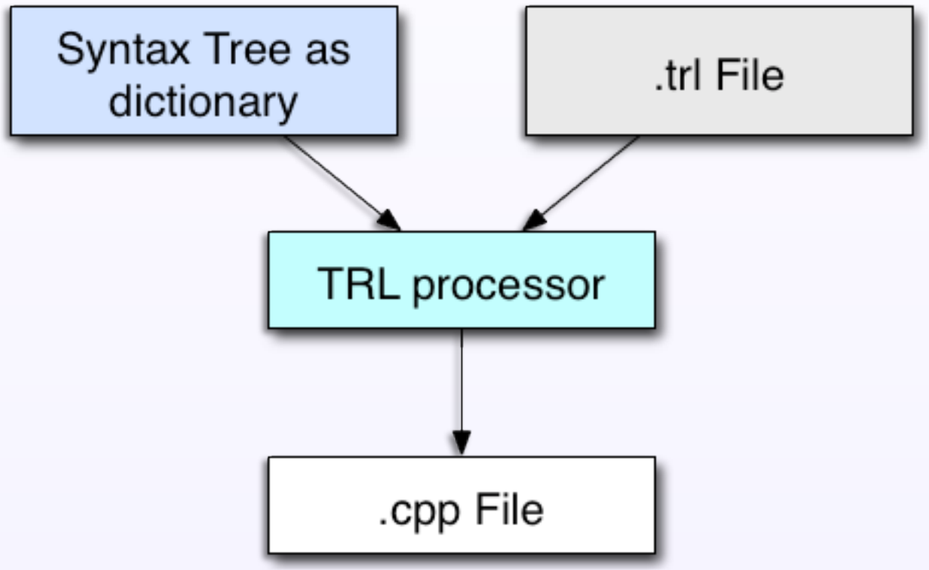
See Also

- CTemplate
- ngTemplate
- Django (HTML focused)
- Jinja (Python)

Fill out a dictionary and feed it to TRL

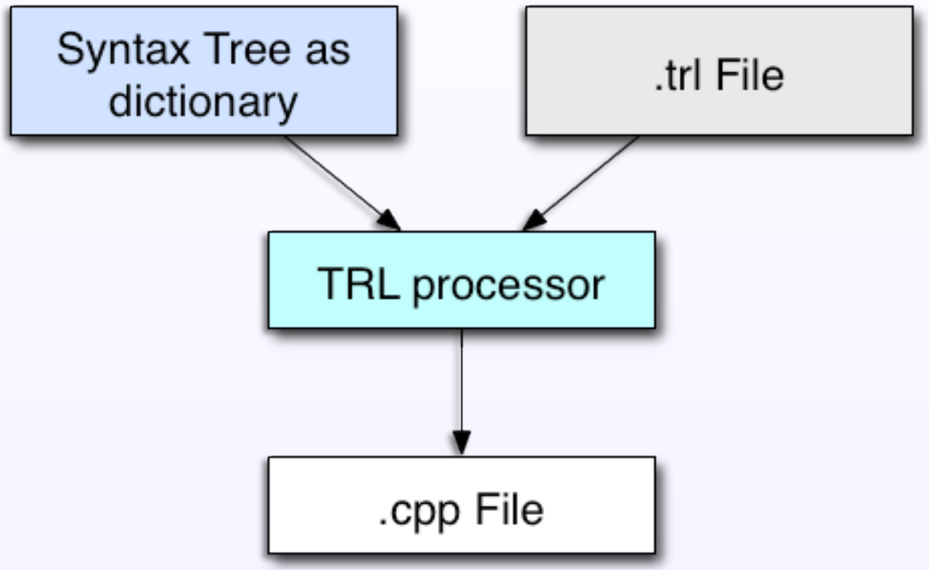


Fill out a dictionary and feed it to TRL



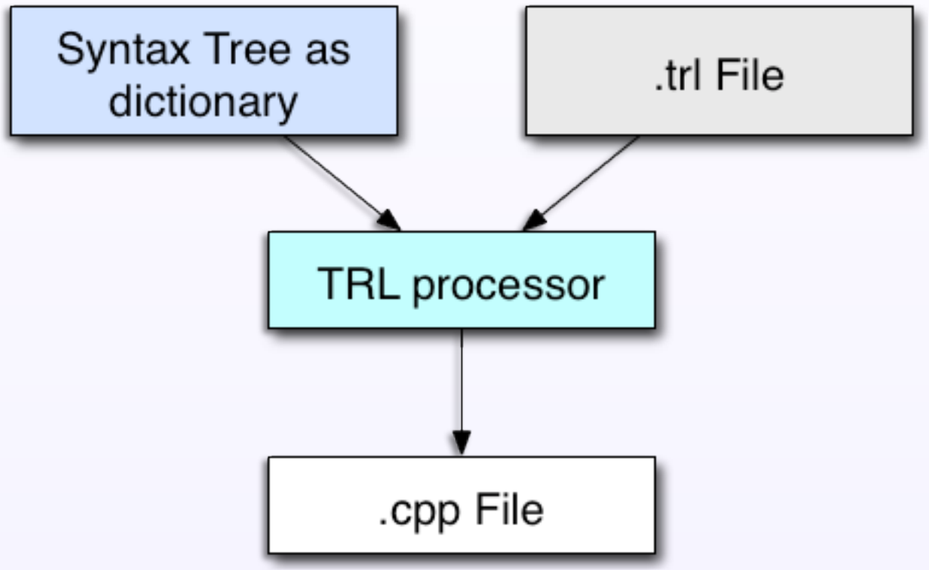

```
{@ foreach f in msg.fields @}
    {@ if f.hasDefault @}
        {{ f.name }} = {{ f.defValue }};
    {@ end if @}
{@ end foreach @}
```

Fill out a dictionary and feed it to TRL




```
{@ foreach f in msg.fields @}
    {@ if f.hasDefault @}
        {{ f.name }} = {{ f.defValue }};
    {@ end if @}
{@ end foreach @}
```

Fill out a dictionary and feed it to TRL




```
{@ foreach f in msg.fields @}
    {@ if f.hasDefault @}
        {{ f.name }} = {{ f.defValue }};
    {@ end if @}
{@ end foreach @}
```

Global Feature addition using TRL

```
{@ end foreach @}
// Virtual encode/decode dispatchers
BOOL {{m.structName}}::Put(teDataChain *pPay
    switch(protocolType) {
        default:
            return FALSE;
        case JAM_PROTOCOL_BINARY_LITERAL:
        {
            teRawBinaryEncoder encoder(pPayl
            return Put(encoder);
        }
        case JAM_PROTOCOL_TEXT_JSON:
        {
            teJSONEncoder encoder(pPayload);
            return Put(encoder);
```

```
177     // Virtual encode/decode dispatchers
178     BOOL {{m.structName}}::Put(teDataChain *pPay
179         switch(protocolType) {
180             default:
181                 return FALSE;
182             case JAM_PROTOCOL_BINARY_COMPRESSED:
183             case JAM_PROTOCOL_BINARY_LITERAL:
184             {
185                 teRawBinaryEncoder encoder(pPayl
186                 return Put(encoder);
187             }
188             case JAM_PROTOCOL_TEXT_JSON_COMPRES
189             case JAM_PROTOCOL_TEXT_JSON:
190             {
191                 teJSONEncoder encoder(pPayload);
```

Development Cycle

- Describe the protocol
- Generate serialization and dispatch
- Send messages
- Receive messages
- Configure routing info

Create a message, fill in data, call send

```
void Checker::OnCaptured(CheckerID capturedBy, JUMP_TYPE how)
{
    CheckerCapturedCredit msg;
    msg.capturedCheckerID = GetID();
    msg.capturedBy = capturedBy;
    msg.jumpType = how;
    JamID destination = GetRouter()->GetCreditManagerID();
    GetRouter()->Send(destination, &msg);
}
```

Create a message, fill in data, call send

```
void Checker::OnCaptured(CheckerID capturedBy, JUMP_TYPE how)
{
    CheckerCapturedCredit msg;
    msg.capturedCheckerID = GetID();
    msg.capturedBy = capturedBy;
    msg.jumpType = how;
    JamID destination = GetRouter()->GetCreditManagerID();
    GetRouter()->Send(destination, &msg);
}
```

Create a message, fill in data, call send

```
void Checker::OnCaptured(CheckerID capturedBy, JUMP_TYPE how)
{
    CheckerCapturedCredit msg;
    msg.capturedCheckerID = GetID();
    msg.capturedBy = capturedBy;
    msg.jumpType = how;
    JamID destination = GetRouter()->GetCreditManagerID();
    GetRouter()->Send(destination, &msg);
}
```

Create a message, fill in data, call send

```
void Checker::OnCaptured(CheckerID capturedBy, JUMP_TYPE how)
{
    CheckerCapturedCredit msg;
    msg.capturedCheckerID = GetID();
    msg.capturedBy = capturedBy;
    msg.jumpType = how;
    JamID destination = GetRouter()->GetCreditManagerID();
    GetRouter()->Send(destination, &msg);
}
```

Create a message, fill in data, call send

```
void Checker::OnCaptured(CheckerID capturedBy, JUMP_TYPE how)
{
    CheckerCapturedCredit msg;
    msg.capturedCheckerID = GetID();
    msg.capturedBy = capturedBy;
    msg.jumpType = how;
    JamID destination = GetRouter()->GetCreditManagerID();
    GetRouter()->Send(destination, &msg);
}
```

Create a message, fill in data, call send

```
void Checker::OnCaptured(CheckerID capturedBy, JUMP_TYPE how)
{
    CheckerCapturedCredit msg;
    msg.capturedCheckerID = GetID();
    msg.capturedBy = capturedBy;
    msg.jumpType = how;
    JamID destination = GetRouter()->GetCreditManagerID();
    GetRouter()->Send(destination, &msg);
}
```

Structs and arrays in messages

```
message GroupUpdate
{
    GroupID group;
    array<.CheckerID> checkers;
};
```

```
    /*** DATA START ***/
    GroupID group;
    vector<CheckerID> checkers;
    /*** DATA STOP ***/
```

```
void GroupService::SendUpdate(GroupID id)
{
    GroupUpdate msg;
    msg.group = id;
    msg.checkers.resize(MAX_GROUP_SIZE);
    // ...
}
```

Structs and arrays in messages

```
message GroupUpdate
{
    GroupID group;
    array<.CheckerID> checkers;
};
```

```
    /*** DATA START ***/
    GroupID group;
    vector<CheckerID> checkers;
    /*** DATA STOP ***/
```

```
void GroupService::SendUpdate(GroupID id)
{
    GroupUpdate msg;
    msg.group = id;
    msg.checkers.resize(MAX_GROUP_SIZE);
    // ...
}
```

Structs and arrays in messages

```
message GroupUpdate
{
    GroupID group;
    array<.CheckerID> checkers;
};
```

```
    /*** DATA START ***/
    GroupID group;
    vector<CheckerID> checkers;
    /*** DATA STOP ***/
```

```
void GroupService::SendUpdate(GroupID id)
{
    GroupUpdate msg;
    msg.group = id;
    msg.checkers.resize(MAX_GROUP_SIZE);
    // ...
}
```

Structs and arrays in messages

```
message GroupUpdate
{
    GroupID group;
    array<.CheckerID> checkers;
};
```

```
    /*** DATA START ***/
    GroupID group;
    vector<CheckerID> checkers;
    /*** DATA STOP ***/
```

```
void GroupService::SendUpdate(GroupID id)
{
    GroupUpdate msg;
    msg.group = id;
    msg.checkers.resize(MAX_GROUP_SIZE);
    // ...
}
```

Structs and arrays in messages

```
message GroupUpdate
{
    GroupID group;
    array<.CheckerID> checkers;
};
```

```
    /*** DATA START ***/
    GroupID group;
    vector<CheckerID> checkers;
    /*** DATA STOP ***/
```

```
void GroupService::SendUpdate(GroupID id)
{
    GroupUpdate msg;
    msg.group = id;
    msg.checkers.resize(MAX_GROUP_SIZE);
    // ...
}
```

Definitions

Definitions

- Message - serialized structure defined in a .jam file

Definitions

- Message - serialized structure defined in a .jam file
- Protocol - a collection of messages

Definitions

- Message - serialized structure defined in a .jam file
- Protocol - a collection of messages
- Service - a module of code that implements message handlers for one or more protocols

Definitions

- Message - serialized structure defined in a .jam file
- Protocol - a collection of messages
- Service - a module of code that implements message handlers for one or more protocols
- Program - can be composed of multiple services

Message Destinations

Message Destinations

```
void MatchService::CreateBoard(u64 width, u64 height) {
    BoardID = GenerateBoard();
    // Send to a known, connected, service
    m_pServer->Send(m_boardServerID, &msg);
}
```

Message Destinations

```
void MatchService::CreateBoard(u64 width, u64 height) {
    BoardID = GenerateBoard();
    // Send to a known, connected, service
    m_pServer->Send(m_boardServerID, &msg);
}

void MatchService::GameOver(u32 gameID, u64 winnerID) {
    msg.gameID = gameID;
    msg.winner = winnerID();
    // Send to a service type, non-specified ID
    m_pServer->Send(JAM_SERVER_STATS_TRACKER, &msg);
}
```

Message Destinations

```
void MatchService::CreateBoard(u64 width, u64 height) {
    BoardID = GenerateBoard();
    // Send to a known, connected, service
    m_pServer->Send(m_boardServerID, &msg);
}

void MatchService::GameOver(u32 gameID, u64 winnerID) {
    msg.gameID = gameID;
    msg.winner = winnerID();
    // Send to a service type, non-specified ID
    m_pServer->Broadcast(JAM_SERVER_STATS_TRACKER, &msg);
}
```

Message Destinations

```
void MatchService::CreateBoard(u64 width, u64 height) {
    BoardID = GenerateBoard();
    // Send to a known, connected, service
    m_pServer->Send(m_boardServerID, &msg);
}

void MatchService::GameOver(u32 gameID, u64 winnerID) {
    msg.gameID = gameID;
    msg.winner = winnerID();
    // Send to a service type, non-specified ID
    m_pServer->Broadcast(JAM_SERVER_STATS_TRACKER, &msg);
}

void Checker::HealChecker(CheckerID toHeal, u32 amount) {
    CheckerHeal msg;
    msg.healedBy = GetID();
    msg.amount = amount;
    // Send a message to a specific object
    m_pServer->Send(toHeal, &msg);
}
```

Message routing by type

```
MatchmakerAddPlayer addMsg;
addMsg.player = GetPlayerID();
addMsg.rank = GetRank();

// No JamID needed, send to any Matchmaker
// May be queued until a Matchmaker is available
m_pService->Send(JAM_SERVER_MATCHMAKER, &addMsg);
```

Send a message and expect a response

```
MatchmakerAddPlayer addMsg;
addMsg.player = GetPlayerID();
addMsg.level = GetLevel();

// Send to any Matchmaker, PlayerAddedHandler
// will be called with response when complete
m_pService->SendRegistered<PlayerAdded>(
    JAM_SERVER_MATCHMAKER, &addMsg
);
```

Send a message and expect a response

```
MatchmakerAddPlayer addMsg;
addMsg.player = GetPlayerID();
addMsg.level = GetLevel();

// Send to any Matchmaker, PlayerAddedHandler
// will be called with response when complete
m_pService->SendRegistered<PlayerAdded>(
    JAM_SERVER_MATCHMAKER, &addMsg
);
```

Send a message to an object

```
void CheckerGroup::ChangeBoards(u32 newBoard)
{
    CheckerChangeBoard msg;
    msg.boardID = newBoard;
    for(int i = 0; i < m_checkers.size(); i++) {
        m_pServer->Send(m_checkers[i]->GetID(), &msg);
    }
}
```

Each object is owned by one server

```
class Checker {
    //...
    CheckerID m_id;
    JamID m_serverID;

    JamID GetServer() {
        return m_serverID;
    }

    CheckerID GetID() {
        return m_id;
    }
    //...
};
```

Each object is owned by one server

```
class Checker {
    //...
    CheckerID m_id;
    JamID m_serverID;

    JamID GetServer() {
        return m_serverID;
    }

    CheckerID GetID() {
        return m_id;
    }
    //...
};
```

Each object is owned by one server

```
class Checker {
    //...
    CheckerID m_id;
    JamID m_serverID;

    JamID GetServer() {
        return m_serverID;
    }

    CheckerID GetID() {
        return m_id;
    }
    //...
};
```

How messages get routed

```
void BoardServer::Send(Checker *pChecker, JamMessage *pMessage)
{
    m_pJamServer->Send(pChecker->GetServer(),
                       pChecker->GetID(),
                       pMessage);
}
```

Development Cycle

- Describe the protocol
- Generate serialization and dispatch
- Send messages
- Receive messages
- Configure routing info

On receipt, look up and dispatch

```
// static callback registered with JAM by protocol ID
// called for each incoming message
void BoardServer::CheckerDispatch(JamLink &link, JamMessage *pMessage)
{
    CheckerID destID = pMessage->GetDestination();
    Checker *pChecker = GetCheckerObject(destID);
    pChecker->QueueMessage(pMessage);
    switch(pMessage->GetProtocolCRC()) {
        case JAMCheckerProtocol_CRC:
            JamCheckerProtocol::Dispatch<Checker>(pMessage, pChecker);
    }
}
```

On receipt, look up and dispatch

```
// static callback registered with JAM by protocol ID
// called for each incoming message
void BoardServer::CheckerDispatch(JamLink &link, JamMessage *pMessage)
{
    CheckerID destID = pMessage->GetDestination();
    Checker *pChecker = GetCheckerObject(destID);
    pChecker->QueueMessage(pMessage);
    switch(pMessage->GetProtocolCRC()) {
        case JAMCheckerProtocol_CRC:
            JamCheckerProtocol::Dispatch<Checker>(pMessage, pChecker);
    }
}
```

On receipt, look up and dispatch

```
// static callback registered with JAM by protocol ID
// called for each incoming message
void BoardServer::CheckerDispatch(JamLink &link, JamMessage *pMessage)
{
    CheckerID destID = pMessage->GetDestination();
    Checker *pChecker = GetCheckerObject(destID);
    pChecker->QueueMessage(pMessage);
    switch(pMessage->GetProtocolCRC()) {
        case JAMCheckerProtocol_CRC:
            JamCheckerProtocol::Dispatch<Checker>(pMessage, pChecker);
    }
}
```

JamLink

```
void BoardServer::CheckerDispatch(JamLink &link,
                                  JamMessage *pMessage)
{
```

Generated Dispatch methods

```
//NOTICE: This is generated code.  DO NOT EDIT!
template<typename HANDLER_T>
static JAM_RESULT Dispatch(JamMessage *pMessage,
                           HANDLER_T *pHandler) {
    switch(pMessage->GetCode()) {
    case JAM_MSG_CheckerHeal:
        result = pHandler->CheckerHealHandler(link,
            (CheckerHeal *)pMessage);
        break;
    // cases for rest of protocol's messages...
```

Generated Dispatch methods

```
//NOTICE: This is generated code.  DO NOT EDIT!
template<typename HANDLER_T>
static JAM_RESULT Dispatch(JamMessage *pMessage,
                           HANDLER_T *pHandler) {
    switch(pMessage->GetCode()) {
    case JAM_MSG_CheckerHeal:
        result = pHandler->CheckerHealHandler(link,
            (CheckerHeal *)pMessage);
        break;
    // cases for rest of protocol's messages...
```

Generated Dispatch methods

```
//NOTICE: This is generated code.  DO NOT EDIT!
template<typename HANDLER_T>
static JAM_RESULT Dispatch(JamMessage *pMessage,
                           HANDLER_T *pHandler) {
    switch(pMessage->GetCode()) {
    case JAM_MSG_CheckerHeal:
        result = pHandler->CheckerHealHandler(link,
            (CheckerHeal *)pMessage);
        break;
    // cases for rest of protocol's messages...
```

Generated message handler prototypes

```
// A message handler prototype is auto-generated for each message
// in the protocol.  #include these declarations in the middle
// of your hand constructed class.
JAM_RESULT CheckerHealHandler(JamLink &link, CheckerHeal *msg);
JAM_RESULT CheckerDamageHandler(JamLink &link, CheckerDamage *msg);
JAM_RESULT CheckerPowerupHandler(JamLink &link, CheckerPowerup *msg);
JAM_RESULT CheckerKingHandler(JamLink &link, CheckerKing *msg);
```

#include this in the middle of a class

Message handler methods

```
JAM_RESULT Checker::CheckerHealHandler(CheckerHeal *pMessage)
{
    m_health += pMessage->amount;
    LOG("Checker %d was healed for %d by checker %d",
        GetID(), pMessage->amount, pMessage->healedBy);
    return JAM_OK;
}
```

Send and Receive

```
void Checker::HealChecker(CheckerID toHeal, u32 amount) {
    CheckerHeal msg;
    msg.healedBy = GetID();
    msg.amount = amount;
    // Send a message to a specific object
    m_pServer->Send(toHeal, &msg);
}

JAM_RESULT Checker::CheckerHealHandler(CheckerHeal *pMessage)
{
    m_health += pMessage->amount;
    LOG("Checker %d was healed for %d by checker %d",
        GetID(), pMessage->amount, pMessage->healedBy);
    return JAM_OK;
}
```

Development Cycle

- Describe the protocol
- Generate serialization and dispatch
- Send messages
- Receive messages
- Configure routing info

Define services

```
void Matchmaker::Configure(JamServer *pServer)
{
    JamRouteConfig &routeConfig = pServer->GetRouteConfig();
    routeConfig.ConfigureInbound<MatchmakerProtocol>(
        this, Matchmaker::DispatchMessage);
    routeConfig.ConfigureOutbound<MatchmakerResponseProtocol>();
}
```

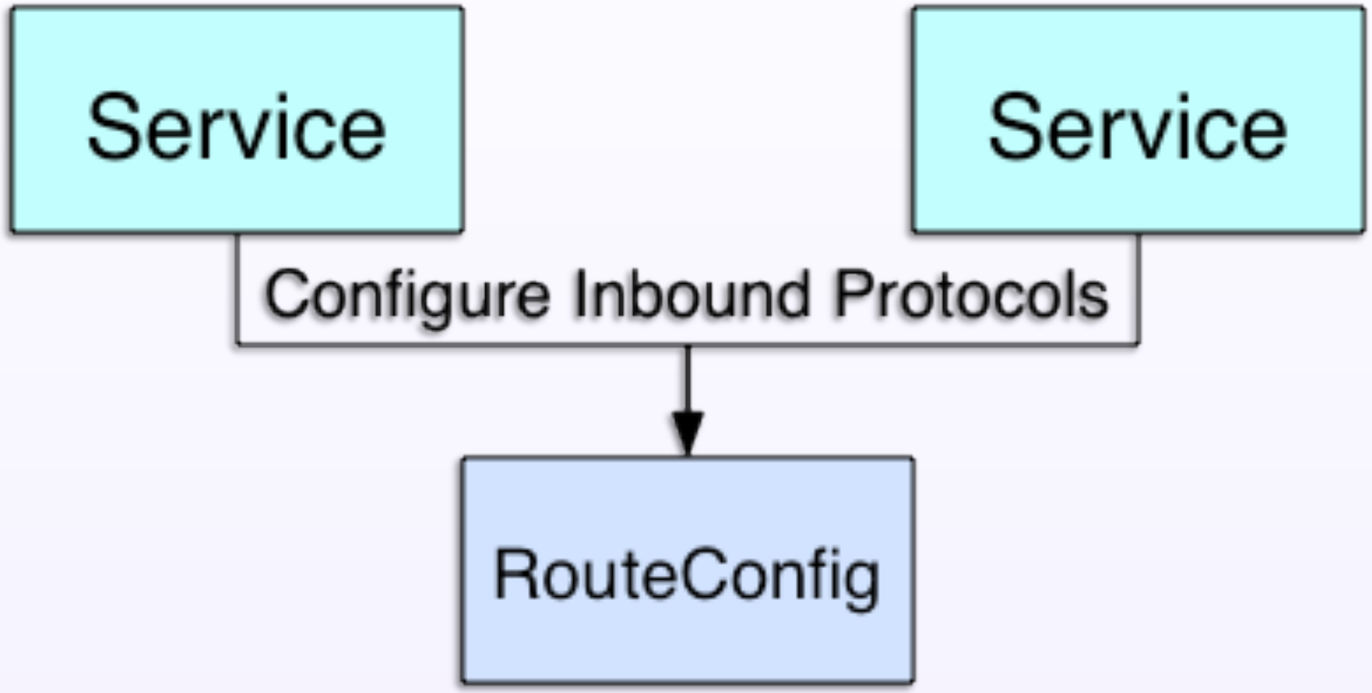
Configure protocols the Matchmaker service sends and receives

Define services

```
void Matchmaker::Configure(JamServer *pServer)
{
    JamRouteConfig &routeConfig = pServer->GetRouteConfig();
    routeConfig.ConfigureInbound<MatchmakerProtocol>(
        this, Matchmaker::DispatchMessage);
    routeConfig.ConfigureOutbound<MatchmakerResponseProtocol>();
}
```

Configure protocols the Matchmaker service sends and receives

RouteConfig maintains a protocol to handler mapping



Handlers have access to sender and other metadata about received messages

```
JAM_RESULT BoardServer::AddPlayerHandler(JamLink &link,
                                         AddPlayer *msg)
{
    LOG("Adding player %s from server %s",
        IDSTR(msg->playerID),
        link.Describe().c_str());
    // Do stuff
    return JAM_OK;
}
```

Handlers have access to sender and other metadata about received messages

```
JAM_RESULT BoardServer::AddPlayerHandler(JamLink &link,
                                         AddPlayer *msg)
{
    LOG("Adding player %s from server %s",
        IDSTR(msg->playerID),
        link.Describe().c_str());
    // Do stuff
    return JAM_OK;
}
```

Coarse and fine-grained queueing and

Race Condition

Receiving via Message Queue

```
void Matchmaker::Configure()
{
    // Messages received at any time are placed into a queue
    routeConfig.ConfigureInbound<MatchmakerProtocol>(
                this, &m_messageQueue);
}

void Matchmaker::Idle()
{
    // Queue is processed in one thread at a known time
    pServer->ProcessQueue(&m_messageQueue, this);
}
```

Receiving via Message Queue

```
void Matchmaker::Configure()
{
    // Messages received at any time are placed into a queue
    routeConfig.ConfigureInbound<MatchmakerProtocol>(
                this, &m_messageQueue);
}

void Matchmaker::Idle()
{
    // Queue is processed in one thread at a known time
    pServer->ProcessQueue(&m_messageQueue, this);
}
```

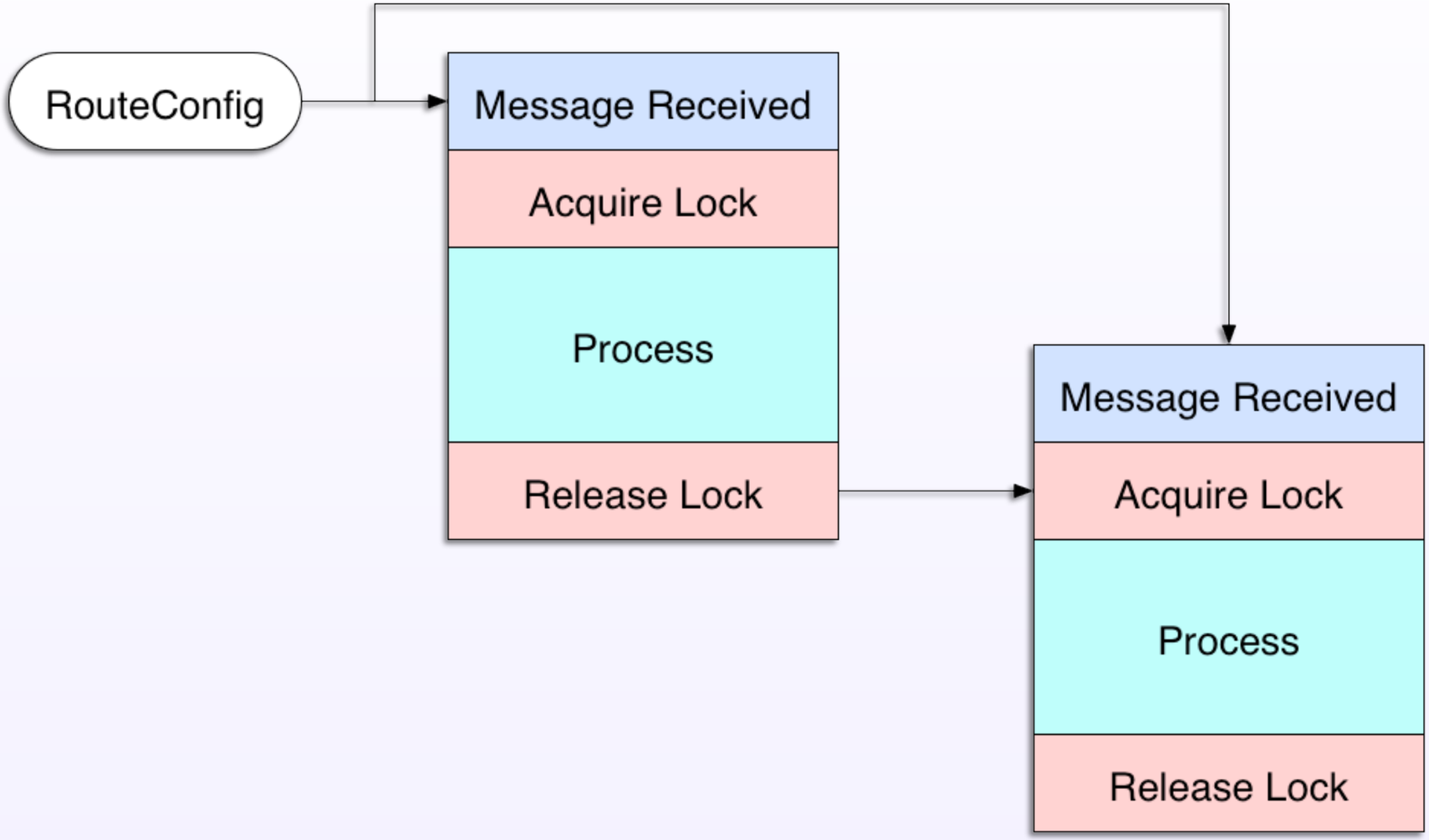
Receiving via Message Queue

```
void Matchmaker::Configure()
{
    // Messages received at any time are placed into a queue
    routeConfig.ConfigureInbound<MatchmakerProtocol>(
                this, &m_messageQueue);
}

void Matchmaker::Idle()
{
    // Queue is processed in one thread at a known time
    pServer->ProcessQueue(&m_messageQueue, this);
}
```

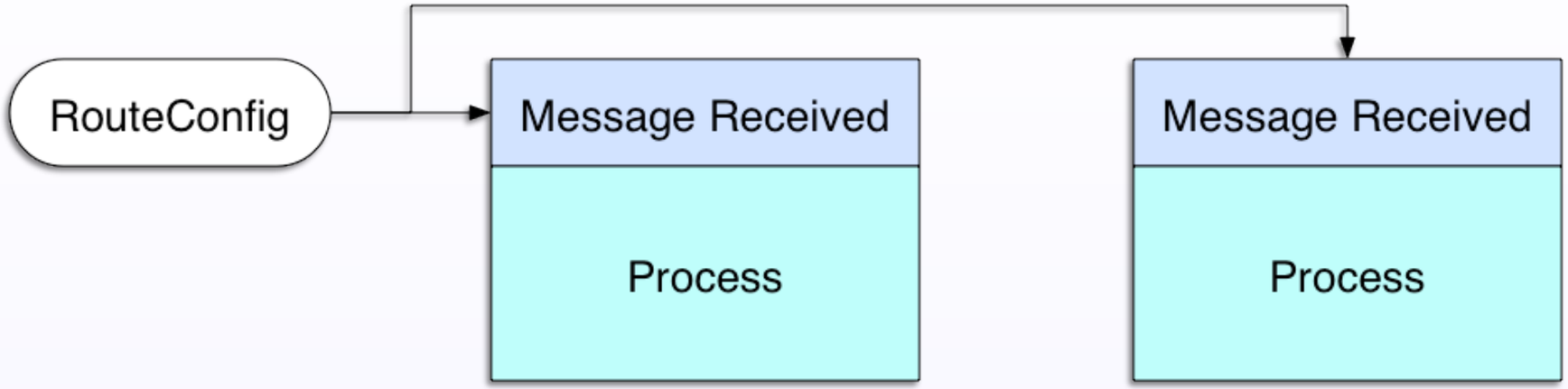
Receiving via Message Queue

```
void Matchmaker::Configure()
{
    // Messages received at any time are placed into a queue
    routeConfig.ConfigureInbound<MatchmakerProtocol>(
                this, &m_messageQueue);
}

void Matchmaker::Idle()
{
    // Queue is processed in one thread at a known time
    pServer->ProcessQueue(&m_messageQueue, this);
}
```

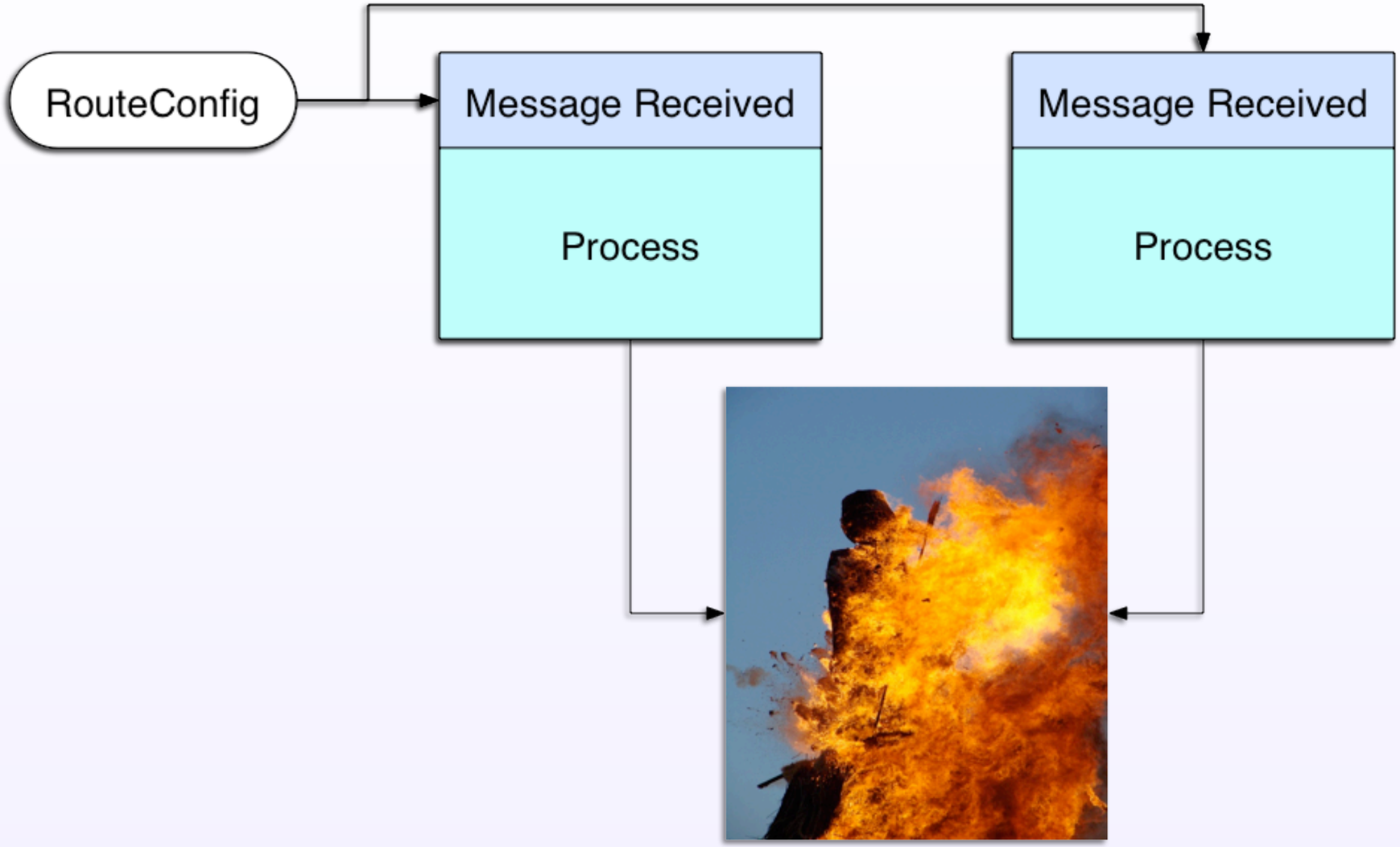
Global lock dispatching



Raw concurrent handlers



Raw concurrent handlers



Lock Policies

```
class MatchmakerLockPolicy
{
    Matchmaker *m_owner;
    void Lock(JamMessage *msg, JamMessageQueue **ppQueue)
    {
        // Adding a player requires a write lock
        if(msg->GetCode() == JAM_MSG_MatchmakerAddPlayer) {
            m_owner->AcquireWriteLock();
        } else {
            m_owner->AcquireReadLock();
        }
    }
    void Unlock(JamMessage *msg) { /* Same logic, release lock */ }
}
```

Lock Policies

```
class MatchmakerLockPolicy
{
    Matchmaker *m_owner;
    void Lock(JamMessage *msg, JamMessageQueue **ppQueue)
    {
        // Adding a player requires a write lock
        if(msg->GetCode() == JAM_MSG_MatchmakerAddPlayer) {
            m_owner->AcquireWriteLock();
        } else {
            m_owner->AcquireReadLock();
        }
    }
    void Unlock(JamMessage *msg) { /* Same logic, release lock */ }
}
```

Incoming messages are refcounted

Incoming messages are refcounted

- Message passed to handler is a refcounted object

Incoming messages are refcounted

- Message passed to handler is a refcounted object
- Possible to retain a message pointer until later

Incoming messages are refcounted

- Message passed to handler is a refcounted object
- Possible to retain a message pointer until later
- Smart pointers are available

Incoming messages are refcounted

- Message passed to handler is a refcounted object
- Possible to retain a message pointer until later
- Smart pointers are available
- Messages contain no pointers to any other objects

Incoming messages are refcounted

- Message passed to handler is a refcounted object
- Possible to retain a message pointer until later
- Smart pointers are available
- Messages contain no pointers to any other objects
- No circular references are possible

CPU And Bandwidth Efficiency

JAM is either efficient or backwards compatible

2004 - Assumed binary compatibility

Negotiation means dead-simple binary serialization most of the time

In some cases, can just memcpy it onto the wire

```
// This message could easily be memcpy'ed onto the wire
class CreateChecker : public JamMessage {
    /**** DATA START ****/
    u32 checkerType;
    u32 owner;
    /**** DATA STOP ****/
    // Code...
};
```

Generated code means easy optimizations

Generated code means easy optimizations

```
_encoder.Put("capturedCheckerID", capturedCheckerID);
_encoder.Put("capturedBy", capturedBy);
_encoder.Put("jumpType", jumpType);
```

Fallback to JSON Serialization

Fallback to JSON Serialization

- Switch to JSON serialization when binary CRC check fails

Fallback to JSON Serialization

- Switch to JSON serialization when binary CRC check fails
- Great for programmers

Fallback to JSON Serialization

- Switch to JSON serialization when binary CRC check fails
- Great for programmers
- Way more expensive (CPU and Bandwidth)

Fallback to JSON Serialization

- Switch to JSON serialization when binary CRC check fails
- Great for programmers
- Way more expensive (CPU and Bandwidth)
- Never allowed on public facing protocols

Fallback to JSON Serialization

- Switch to JSON serialization when binary CRC check fails
- Great for programmers
- Way more expensive (CPU and Bandwidth)
- Never allowed on public facing protocols
- Even internally it's sometimes unreasonable

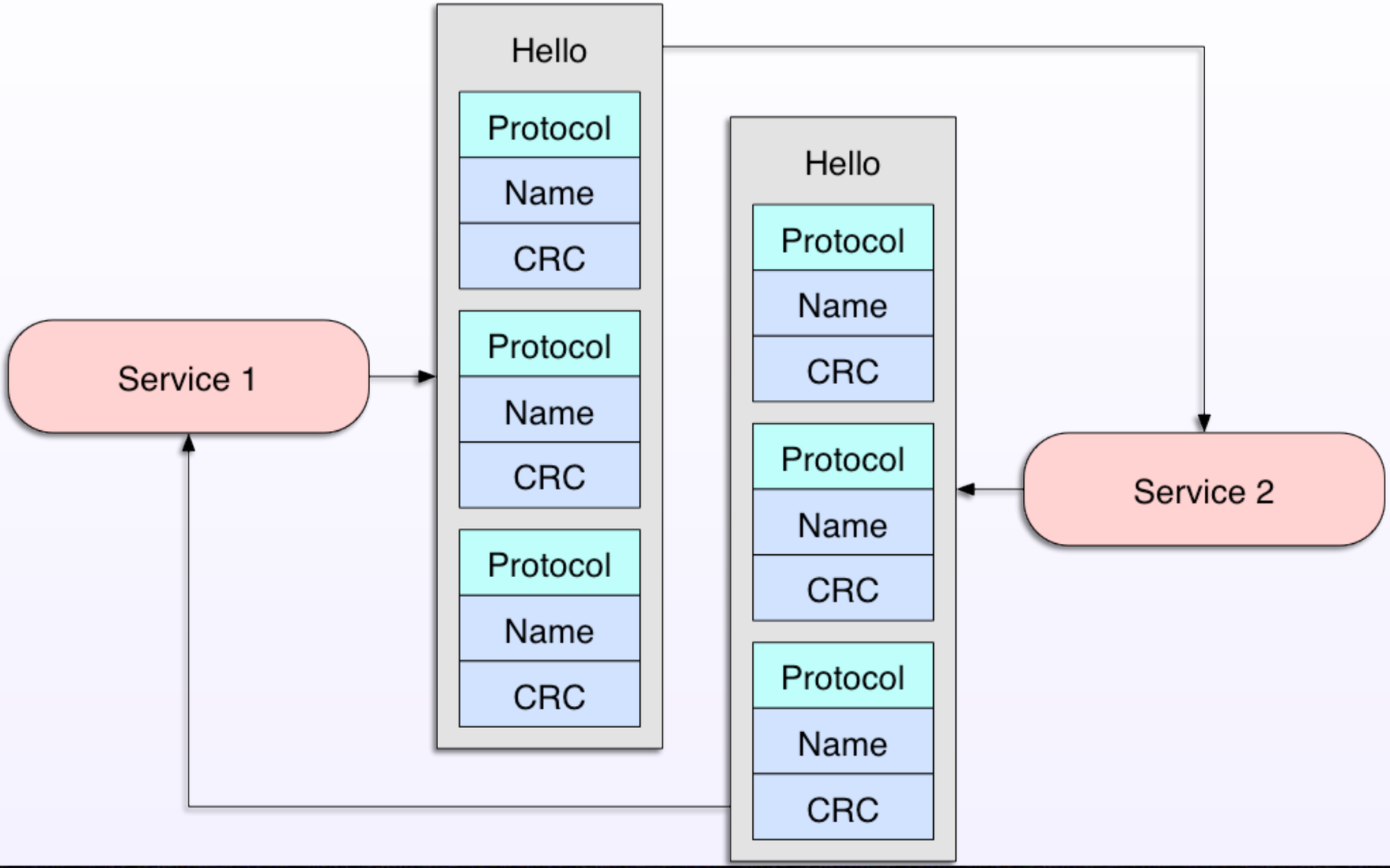
Fallback to JSON Serialization

- Switch to JSON serialization when binary CRC check fails
- Great for programmers
- Way more expensive (CPU and Bandwidth)
- Never allowed on public facing protocols
- Even internally it's sometimes unreasonable

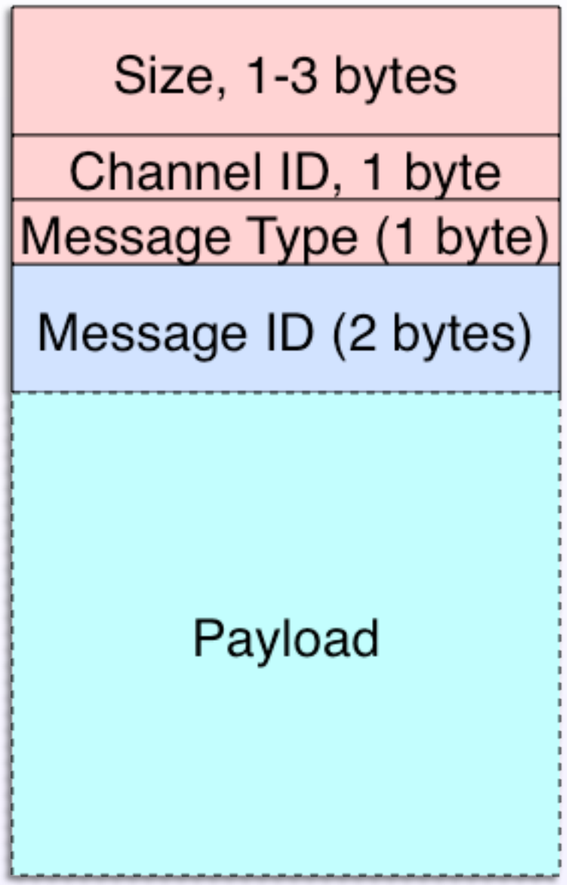
JSON


```
{
    "_msgID":10,
    "type":6,
    "error":0,
    "desc":{
        "m_id":"T2R00S40.00E14815726P10987H127.0.0.1:14001",
        "m_host":"127.0.0.1",
        "m_partitionID":0,
        "m_configID":0,
        "m_buildNum":0,
        "m_type":40,
        "m_subType":0
    }
}
```

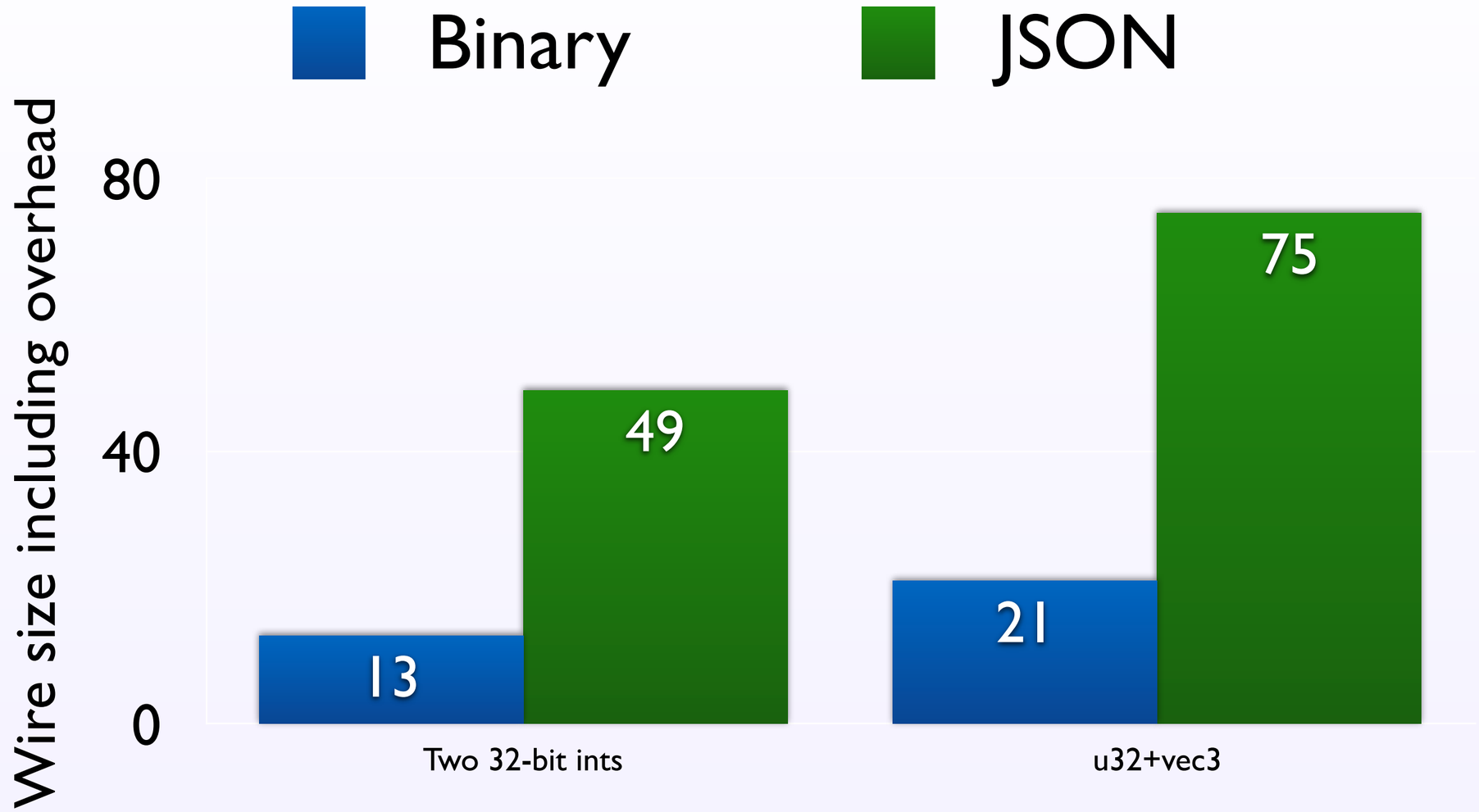
Protocol Negotiation



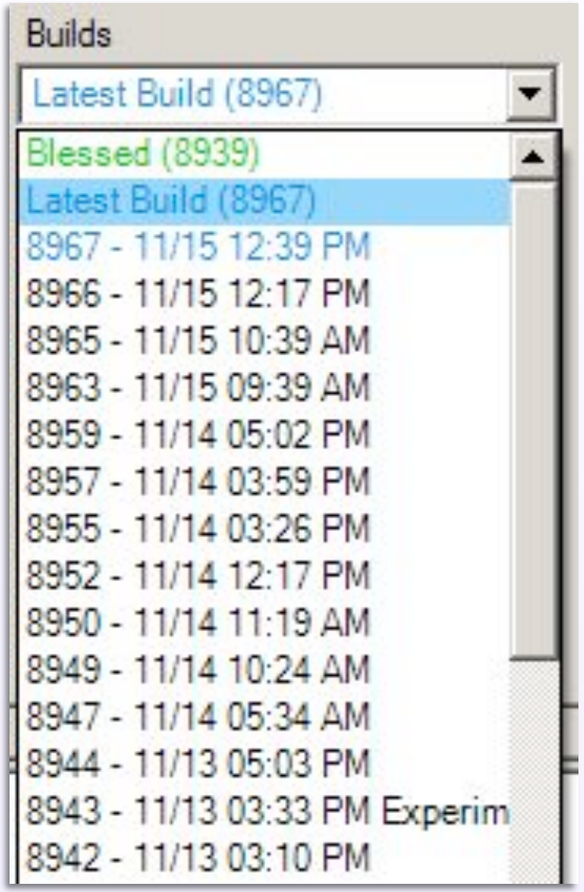
Message overhead



JSON vs. Binary performance



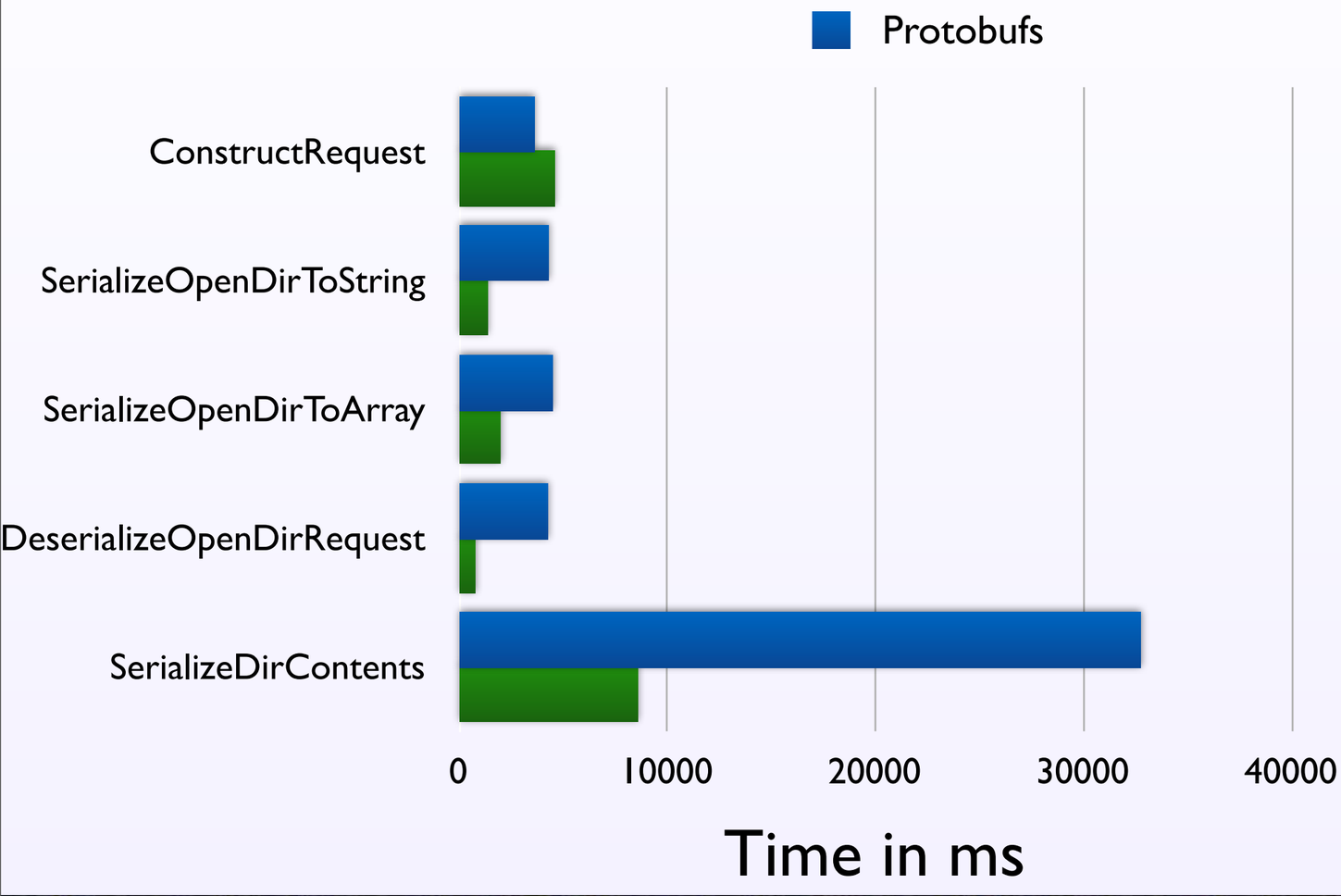
Still highly successful - some network tools run on old versions frequently

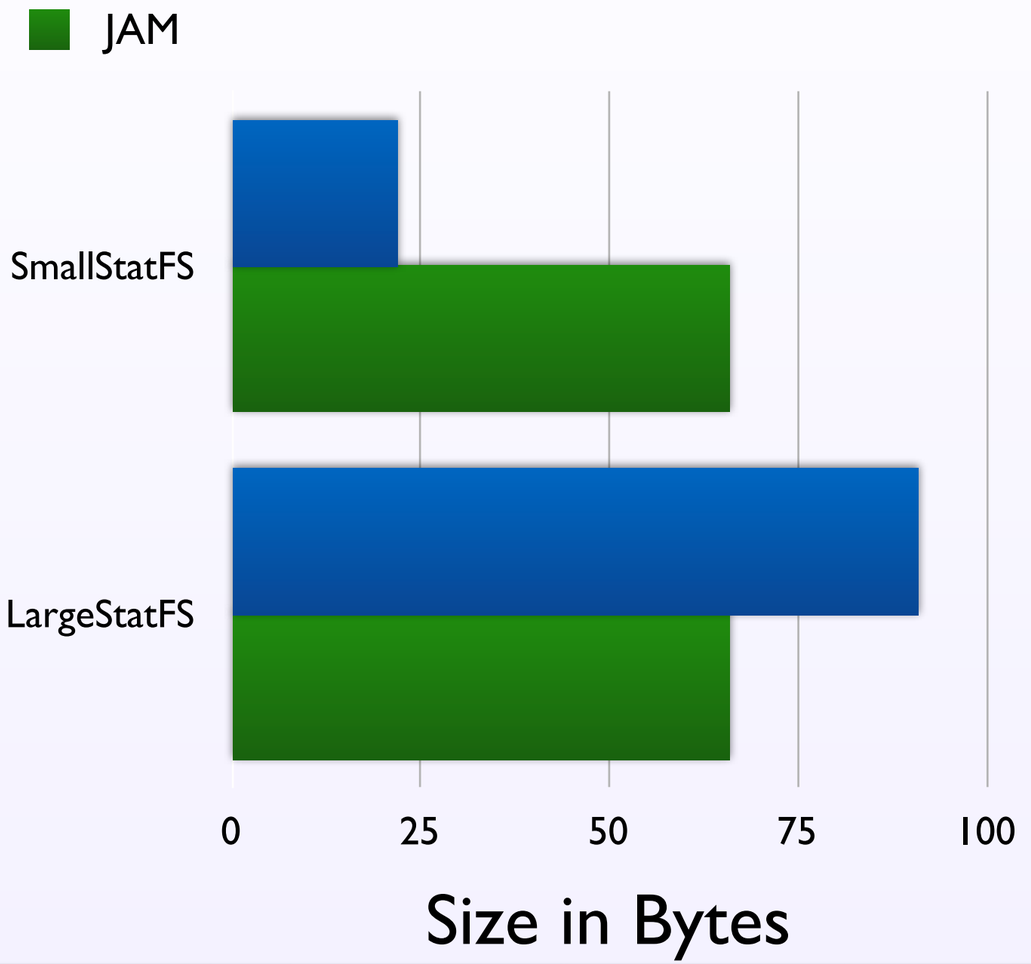


Google's Protocol Buffers and

For both speed AND inter-version compatibility, there are better choices

protobufs sometimes wins on bandwidth, but JAM is faster





Writing our own gives us ultimate control over everything

Writing our own gives us ultimate control over everything

- Automated serialization

Writing our own gives us ultimate control over everything

- Automated serialization
- Easy yet flexible message dispatching

Writing our own gives us ultimate control over everything

- Automated serialization
- Easy yet flexible message dispatching
- High performance

Writing our own gives us ultimate control over everything

- Automated serialization
- Easy yet flexible message dispatching
- High performance
- Inter-version compatibility

Writing our own gives us ultimate control over everything

- Automated serialization
- Easy yet flexible message dispatching
- High performance
- Inter-version compatibility
- Less tedium = more awesome

Thanks! Questions?

Joe Rumsey
jrumsey@blizzard.com
Twitter: @joerumz

Thanks! Questions?

Joe Rumsey
jrumsey@blizzard.com
Twitter: @joerumz

Disclaimer: Blizzard is not really making World of Checkers