Efficient Session Type Guided Distributed Interaction

KC Sivaramakrishnan, Karthik Nagaraj, Lukasz Ziarek, Patrick Eugster

Purdue University

Motivation

Motivation

- Building distributed systems is difficult

Motivation

- Building distributed systems is difficult
- Simplicity over performance and expressivity

Motivation

- Building distributed systems is difficult
- Simplicity over performance and expressivity
- Java Remote Method Invocation (RMI)

Motivation

- Building distributed systems is difficult
- Simplicity over performance and expressivity
- Java Remote Method Invocation (RMI)
  - Invoke remote methods on remote objects through proxies

Motivation

- Building distributed systems is difficult
- Simplicity over performance and expressivity
- Java Remote Method Invocation (RMI)
  - Invoke remote methods on remote objects through proxies
  - Hides network complexities from the programmer

Motivation

- Building distributed systems is difficult
- Simplicity over performance and expressivity
- Java Remote Method Invocation (RMI)
  - Invoke remote methods on remote objects through proxies
  - Hides network complexities from the programmer
  - Limitations

Motivation

- Building distributed systems is difficult
- Simplicity over performance and expressivity
- Java Remote Method Invocation (RMI)
  - Invoke remote methods on remote objects through proxies
  - Hides network complexities from the programmer
  - Limitations
    - synchronous and sequential

Motivation

- Building distributed systems is difficult
- Simplicity over performance and expressivity
- Java Remote Method Invocation (RMI)
    - Invoke remote methods on remote objects through proxies
    - Hides network complexities from the programmer
    - Limitations
        - synchronous and sequential
            - Overuse can lead to poor performance

Motivation

- Building distributed systems is difficult
- Simplicity over performance and expressivity
- Java Remote Method Invocation (RMI)
  - Invoke remote methods on remote objects through proxies
  - Hides network complexities from the programmer
  - Limitations
    - synchronous and sequential
      - Overuse can lead to poor performance
    - cannot reason about sessions

Invitation Example - The Setting

Invitation Example - The Setting

- Bob wants to throw a party to colleagues

Invitation Example - The Setting

- Bob wants to throw a party to colleagues
- Utilize a social networking API for accessing friends' data

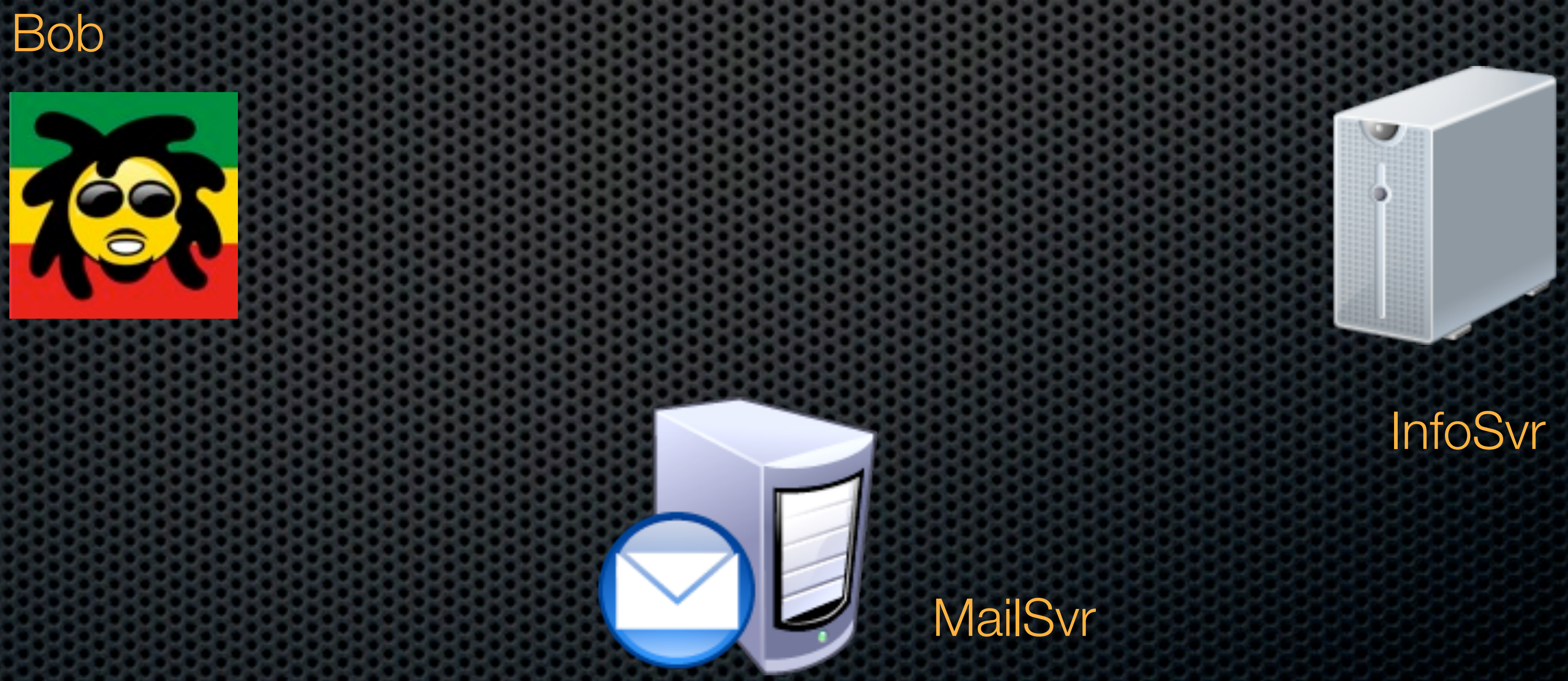
Invitation Example - The Setting

- Bob wants to throw a party to colleagues
- Utilize a social networking API for accessing friends' data
- Social networking API is implemented in Java RMI

Invitation Example - The Setting

- Bob wants to throw a party to colleagues
- Utilize a social networking API for accessing friends' data
- Social networking API is implemented in Java RMI
- Email invitation is sent for chosen colleagues

Invitation Example - RMI Style



Invitation Example - RMI Style

```
void invite_coworkers() {
  Event evt = me.createEvent(“party”,“June 7th, 2010);
  Employer myEmp = me.getEmployer();
  Location myLoc = me.getLocation();
  for (Member friend : me.getFriends()) {
    if (myEmp.equals(friend.getEmployer()) &&
        myLoc.equals(friend.getLocation()) &&
        User.approve(friend)) {
         mailSvr.sendMail(friend.getEmailId(),evt);
    }
  }
```

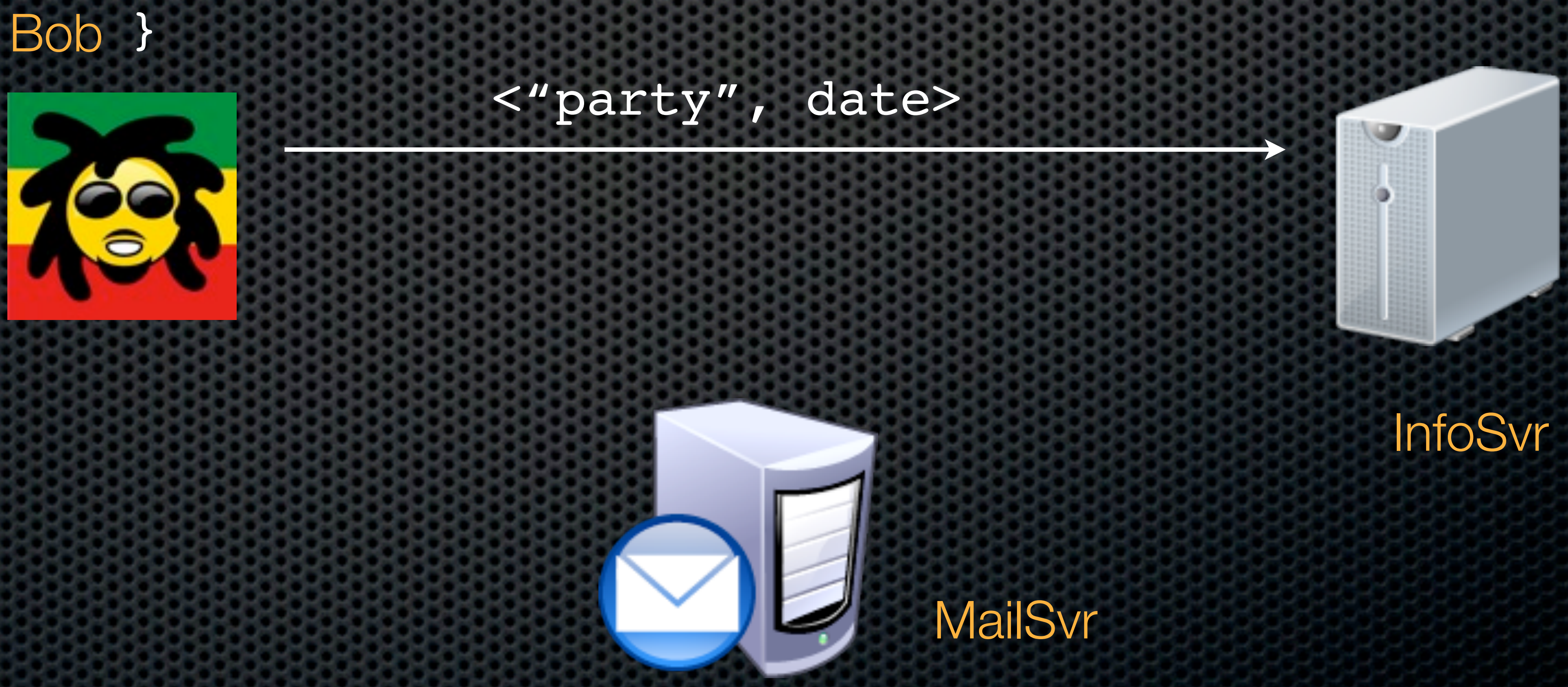
Bob

InfoSvr

MailSvr

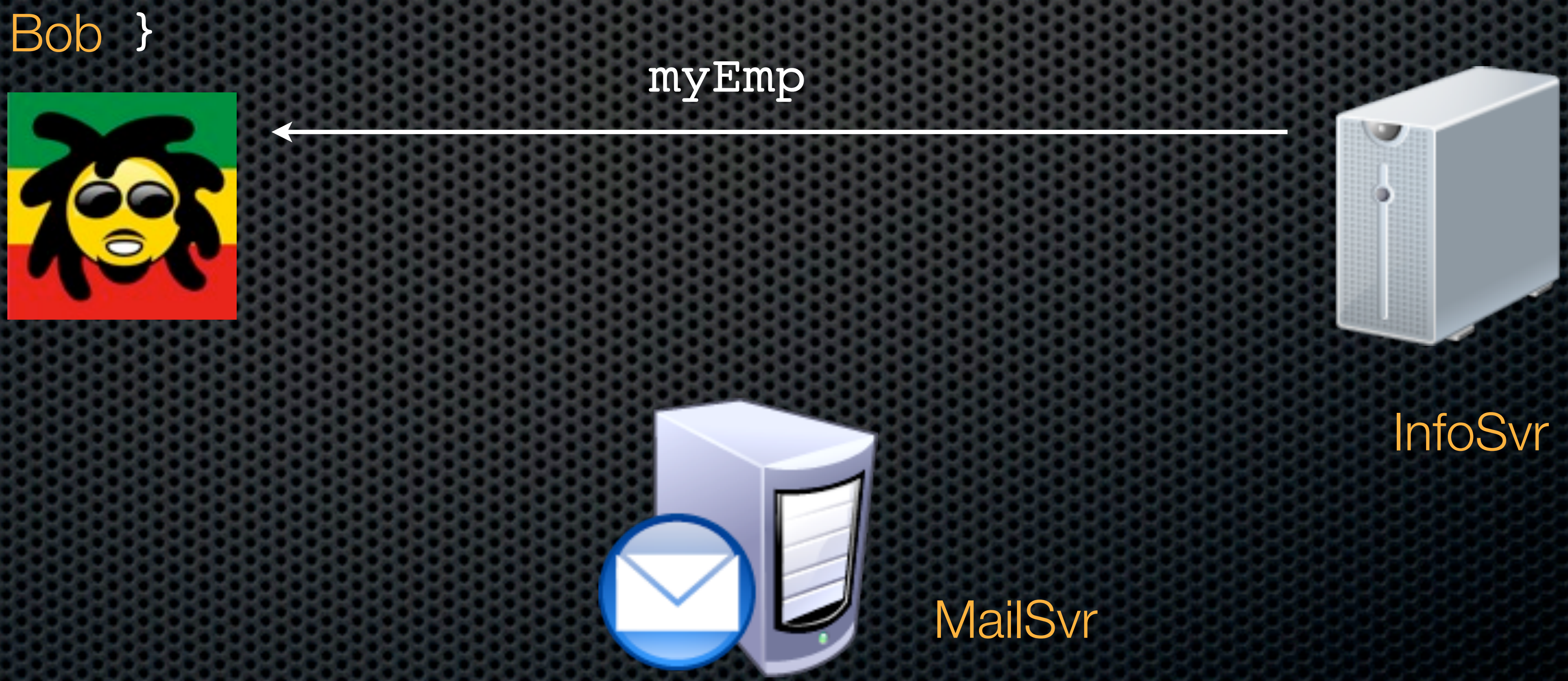
Invitation Example - RMI Style

```
void invite_coworkers() {
  Event evt = me.createEvent(“party”,“June 7th, 2010);
  Employer myEmp = me.getEmployer();
  Location myLoc = me.getLocation();
  for (Member friend : me.getFriends()) {
    if (myEmp.equals(friend.getEmployer()) &&
        myLoc.equals(friend.getLocation()) &&
        User.approve(friend)) {
         mailSvr.sendMail(friend.getEmailId(),evt);
    }
  }
```



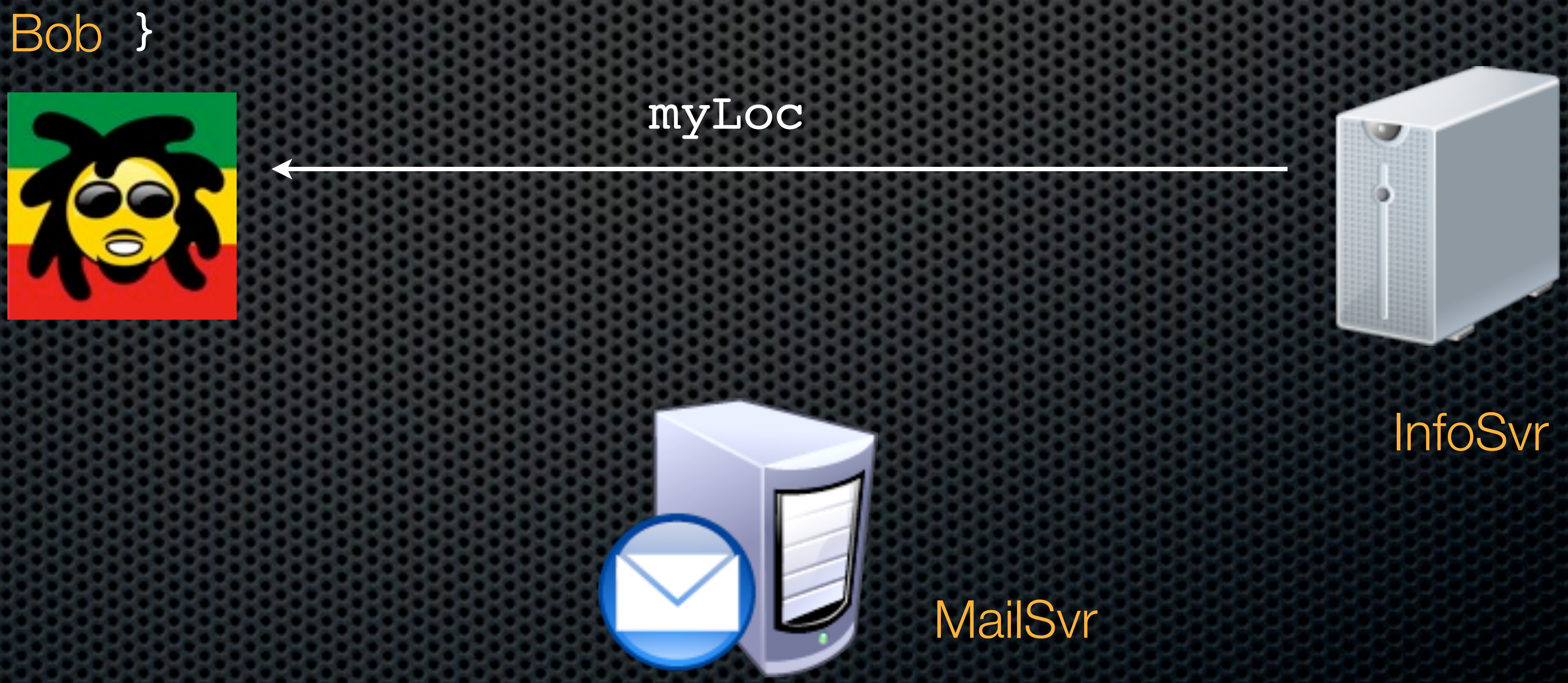
Invitation Example - RMI Style

```
void invite_coworkers() {
  Event evt = me.createEvent(“party”,“June 7th, 2010);
  Employer myEmp = me.getEmployer();
  Location myLoc = me.getLocation();
  for (Member friend : me.getFriends()) {
    if (myEmp.equals(friend.getEmployer()) &&
        myLoc.equals(friend.getLocation()) &&
        User.approve(friend)) {
         mailSvr.sendMail(friend.getEmailId(),evt);
    }
  }
```



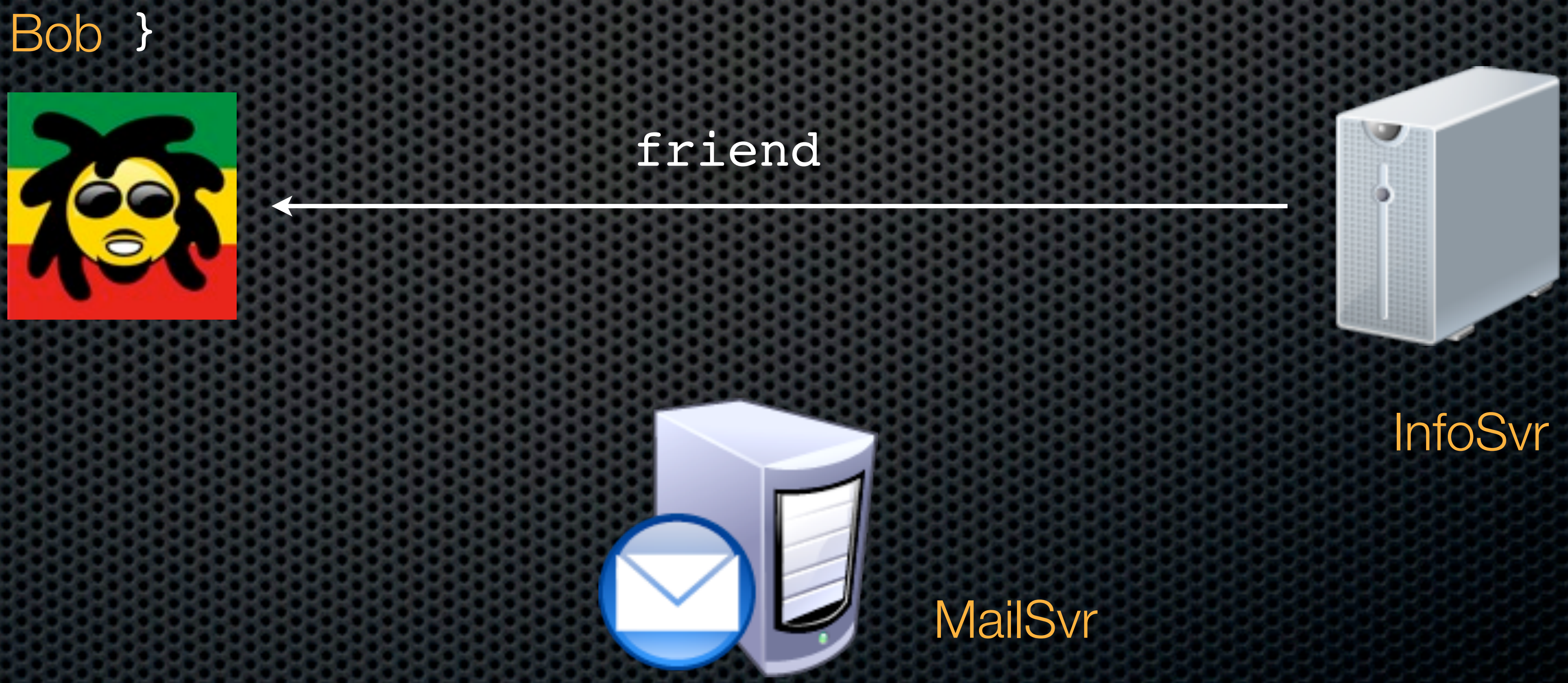
Invitation Example - RMI Style

```
void invite_coworkers() {
  Event evt = me.createEvent(“party”,“June 7th, 2010);
  Employer myEmp = me.getEmployer();
  Location myLoc = me.getLocation();
  for (Member friend : me.getFriends()) {
    if (myEmp.equals(friend.getEmployer()) &&
        myLoc.equals(friend.getLocation()) &&
        User.approve(friend)) {
         mailSvr.sendMail(friend.getEmailId(),evt);
    }
  }
```



Invitation Example - RMI Style

```
void invite_coworkers() {
  Event evt = me.createEvent(“party”,“June 7th, 2010);
  Employer myEmp = me.getEmployer();
  Location myLoc = me.getLocation();
  for (Member friend : me.getFriends()) {
    if (myEmp.equals(friend.getEmployer()) &&
        myLoc.equals(friend.getLocation()) &&
        User.approve(friend)) {
         mailSvr.sendMail(friend.getEmailId(),evt);
    }
  }
```



Invitation Example - RMI Style

```
void invite_coworkers() {
  Event evt = me.createEvent(“party”,“June 7th, 2010);
  Employer myEmp = me.getEmployer();
  Location myLoc = me.getLocation();
  for (Member friend : me.getFriends()) {
    if (myEmp.equals(friend.getEmployer()) &&
        myLoc.equals(friend.getLocation()) &&
        User.approve(friend)) {
         mailSvr.sendMail(friend.getEmailId(),evt);
    }
  }
```

Bob

InfoSvr

MailSvr

Invitation Example - RMI Style

```
void invite_coworkers() {
  Event evt = me.createEvent(“party”,“June 7th, 2010);
  Employer myEmp = me.getEmployer();
  Location myLoc = me.getLocation();
  for (Member friend : me.getFriends()) {
    if (myEmp.equals(friend.getEmployer()) &&
        myLoc.equals(friend.getLocation()) &&
        User.approve(friend)) {
         mailSvr.sendMail(friend.getEmailId(),evt);
    }
  }
```

Invitation Example - RMI Style

```
void invite_coworkers() {
  Event evt = me.createEvent(“party”,“June 7th, 2010);
  Employer myEmp = me.getEmployer();
  Location myLoc = me.getLocation();
  for (Member friend : me.getFriends()) {
    if (myEmp.equals(friend.getEmployer()) &&
        myLoc.equals(friend.getLocation()) &&
        User.approve(friend)) {
         mailSvr.sendMail(friend.getEmailId(),evt);
    }
  }
```

✓

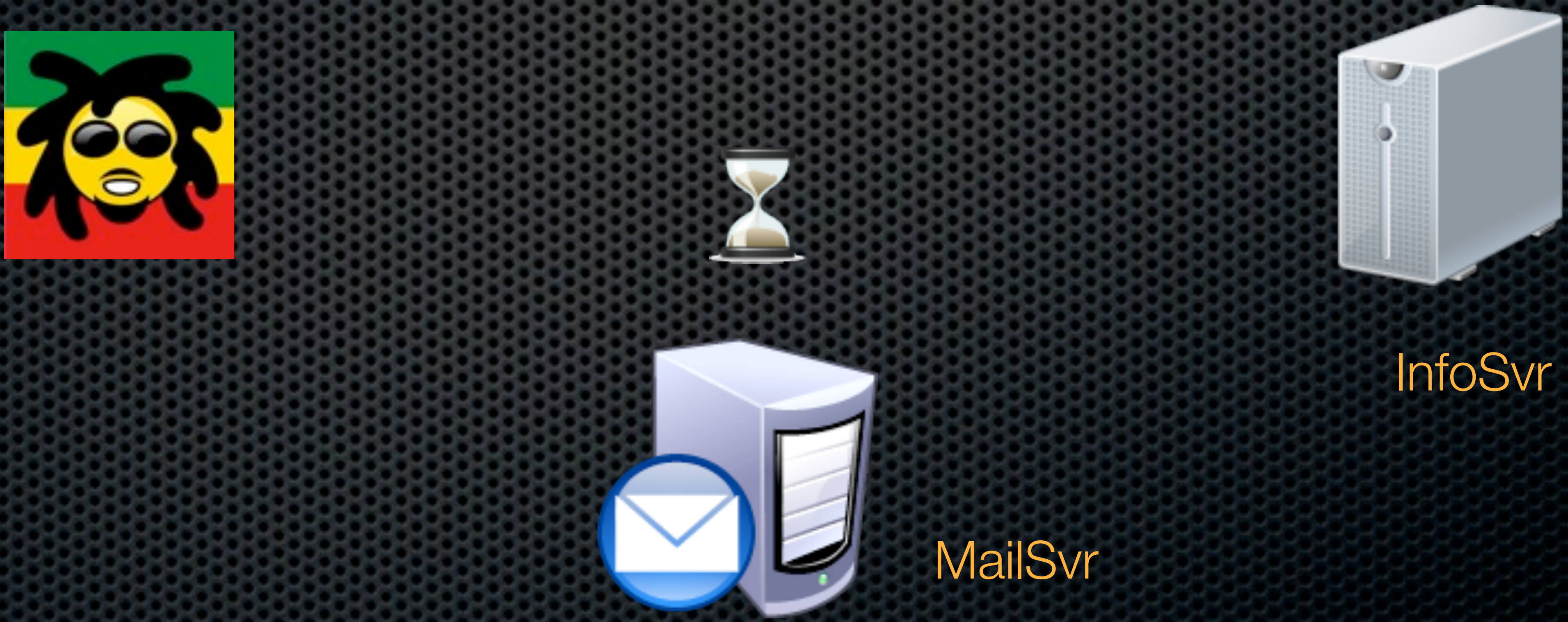
Invitation Example - RMI Style

```
void invite_coworkers() {
  Event evt = me.createEvent(“party”,“June 7th, 2010);
  Employer myEmp = me.getEmployer();
  Location myLoc = me.getLocation();
  for (Member friend : me.getFriends()) {
    if (myEmp.equals(friend.getEmployer()) &&
        myLoc.equals(friend.getLocation()) &&
        User.approve(friend)) {
         mailSvr.sendMail(friend.getEmailId(),evt);
    }
  }
```

Invitation Example - RMI Style

```
void invite_coworkers() {
  Event evt = me.createEvent(“party”,“June 7th, 2010);
  Employer myEmp = me.getEmployer();
  Location myLoc = me.getLocation();
  for (Member friend : me.getFriends()) {
    if (myEmp.equals(friend.getEmployer()) &&
        myLoc.equals(friend.getLocation()) &&
        User.approve(friend)) {
         mailSvr.sendMail(friend.getEmailId(),evt);
    }
  }
```

Bob



Reducing Remote Calls

Reducing Remote Calls

- Export entire function call to `InfoSvr`

Reducing Remote Calls

- Export entire function call to `InfoSvr`
    - Not possible due to user approval process

Reducing Remote Calls

- Export entire function call to `InfoSvr`
  - Not possible due to user approval process
- Remote facade pattern

Reducing Remote Calls

- Export entire function call to `InfoSvr`
  - Not possible due to user approval process
- Remote facade pattern
  - Specialized remote method for each client access pattern

Reducing Remote Calls

- Export entire function call to `InfoSvr`
  - Not possible due to user approval process
- Remote facade pattern
  - Specialized remote method for each client access pattern
  - Server code needs to be changed

Reducing Remote Calls

- Export entire function call to `InfoSvr`
  - Not possible due to user approval process
- Remote facade pattern
  - Specialized remote method for each client access pattern
  - Server code needs to be changed
- Data transfer object

Reducing Remote Calls

- Export entire function call to `InfoSvr`
  - Not possible due to user approval process
- Remote facade pattern
  - Specialized remote method for each client access pattern
  - Server code needs to be changed
- Data transfer object
  - Single coarse grained data transfer instead of multiple fine grained transfers

Reducing Remote Calls

- Export entire function call to `InfoSvr`
  - Not possible due to user approval process
- Remote facade pattern
  - Specialized remote method for each client access pattern
  - Server code needs to be changed
- Data transfer object
  - Single coarse grained data transfer instead of multiple fine grained transfers
  - Over-approximation

Goals

Goals

How do we...

Goals

How do we...

- Automatically reduce remote communication actions

Goals

How do we...

- Automatically reduce remote communication actions
- Optimize multi-party communication

Goals

How do we...

- Automatically reduce remote communication actions
- Optimize multi-party communication

while

Goals

How do we...

- Automatically reduce remote communication actions
- Optimize multi-party communication

while

- preserving semantics of remote execution

Goals

How do we...

- Automatically reduce remote communication actions
- Optimize multi-party communication

while

- preserving semantics of remote execution
- not imposing substantial runtime overheads

Session Type

Session Type

- Abstraction to precisely describe communication protocols

Session Type

- Abstraction to precisely describe communication protocols
    - Typed messages

Session Type

- Abstraction to precisely describe communication protocols
  - Typed messages
  - Ordered

Session Type

- Abstraction to precisely describe communication protocols
  - Typed messages
  - Ordered
  - Explicit control flow information through label selection and recursive types

Session Type

- Abstraction to precisely describe communication protocols
  - Typed messages
  - Ordered
  - Explicit control flow information through label selection and recursive types
- Multiparty Asynchronous Session Types [Honda et al. POPL '08]

Session Type

- Abstraction to precisely describe communication protocols
  - Typed messages
  - Ordered
  - Explicit control flow information through label selection and recursive types
- Multiparty Asynchronous Session Types [Honda et al. POPL '08]
- bi-party session types for Java [Hu et al. ECOOP '08]

Session Type Guided Optimization

Session Type Guided Optimization

- Session types for protocol optimization

Session Type Guided Optimization

- Session types for protocol optimization
  - Utilize type and control flow information for direct optimization

Session Type Guided Optimization

- Session types for protocol optimization
  - Utilize type and control flow information for direct optimization
  - Program transformation through session type guided data flow analysis

Session Type Guided Optimization

- Session types for protocol optimization
  - Utilize type and control flow information for direct optimization
  - Program transformation through session type guided data flow analysis
- Java extension for multi-party session types

Session Type Guided Optimization

- Session types for protocol optimization
  - Utilize type and control flow information for direct optimization
  - Program transformation through session type guided data flow analysis
- Java extension for multi-party session types
  - Language extension

Session Type Guided Optimization

- Session types for protocol optimization
  - Utilize type and control flow information for direct optimization
  - Program transformation through session type guided data flow analysis
- Java extension for multi-party session types
  - Language extension
  - Compiler and runtime framework

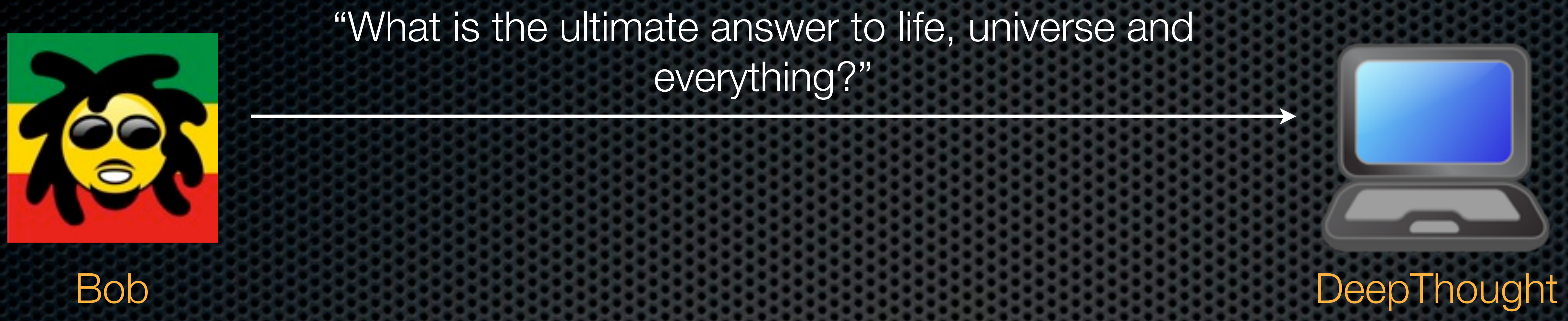
Simple Example

Simple Example

Bob

DeepThought

Simple Example

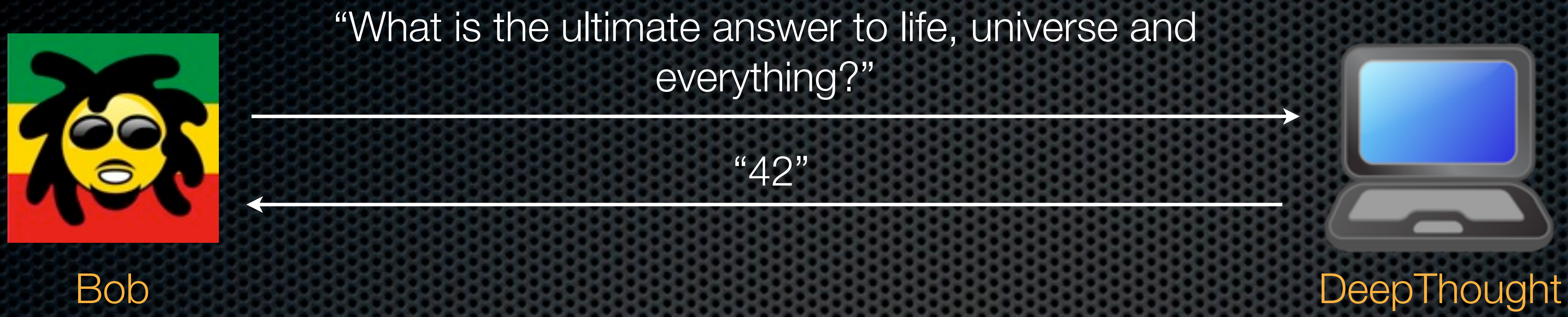


“What is the ultimate answer to life, universe and everything?”

Bob

DeepThought

Simple Example



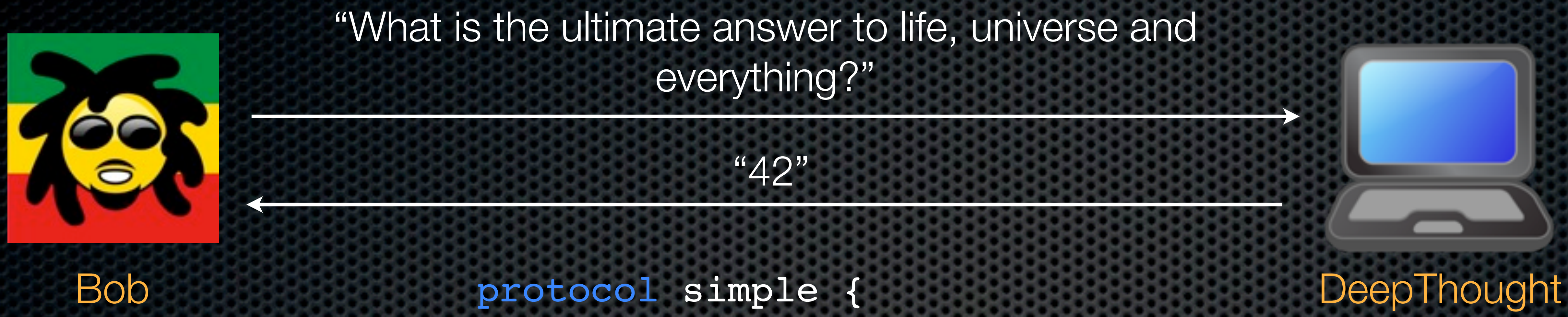
“What is the ultimate answer to life, universe and everything?”

“42”

Bob

DeepThought

Simple Example



“What is the ultimate answer to life, universe and everything?”

“42”

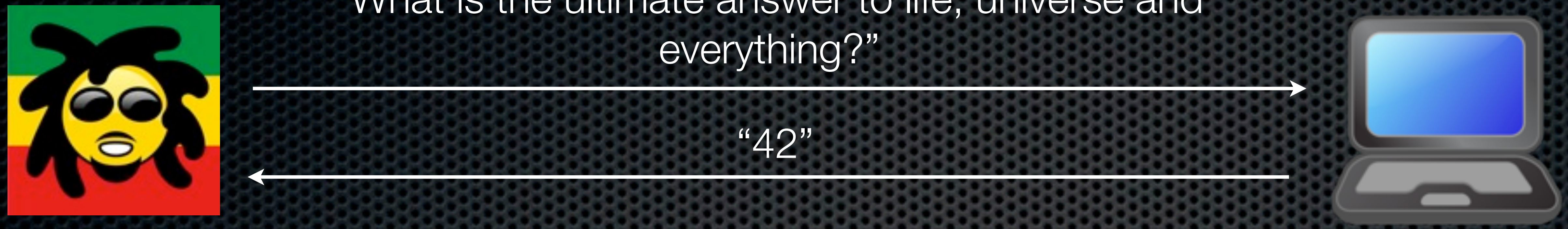
Bob

DeepThought

```
protocol simple {
  participants Bob, DeepThought;
  Bob: begin;
  Bob->DeepThought:<string>;
  DeepThought->Bob:<string>;
}
```

Global session type

Simple Example



“What is the ultimate answer to life, universe and everything?”

“42”

Bob

DeepThought

```
protocol simple {
  participants Bob, DeepThought;
  Bob: begin;
  Bob->DeepThought:<string>;
  DeepThought->Bob:<string>;
}
```

Global session type

```
protocol simple@Bob {
  !begin;
  DeepThought:!<string>;
  DeepThought:?<string>;
}
```

```
protocol simple@DeepThought {
  Bob:?begin;
  Bob:?<string>;
  Bob:!<string>;
}
```

Local session types

Session Implementation

Session Implementation

- Programmer implements the participant with Java extension for session type

Session Implementation

- Programmer implements the participant with Java extension for session type
- Session implementation is statically verified for conformance with local session type

Session Implementation

- Programmer implements the participant with Java extension for session type
- Session implementation is statically verified for conformance with local session type
- Runtime converts sends, receives and control flow actions to network transfers

Session Implementation

- Programmer implements the participant with Java extension for session type
- Session implementation is statically verified for conformance with local session type
- Runtime converts sends, receives and control flow actions to network transfers
- Exceptions are raised upon node and network failures

Invitation Example - Session Type

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer>;
  InfoSvr->Bob: <Location>;
  InfoSvr:
    [InfoSvr->Bob: <Member>;
     InfoSvr->Bob: <Employer>;
     InfoSvr->Bob: <Location>;
     InfoSvr->Bob: <EmailAddr>;
     Bob:
     {INVITE: Bob->MailSvr: <EmailAddr,Event>,
      NOOP:}
    ]*
}
```

Invitation Example - Session Type

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer>;
  InfoSvr->Bob: <Location>;
  InfoSvr:
    [InfoSvr->Bob: <Member>;
     InfoSvr->Bob: <Employer>;
     InfoSvr->Bob: <Location>;
     InfoSvr->Bob: <EmailAddr>;
     Bob:
     {INVITE: Bob->MailSvr: <EmailAddr,Event>,
      NOOP:}
    ]*
}
```

Recursive type

Invitation Example - Session Type

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer>;
  InfoSvr->Bob: <Location>;
  InfoSvr:
    [InfoSvr->Bob: <Member>;
     InfoSvr->Bob: <Employer>;
     InfoSvr->Bob: <Location>;
     InfoSvr->Bob: <EmailAddr>;
     Bob:
     {INVITE: Bob->MailSvr: <EmailAddr,Event>,
      NOOP:}
    ]*
}
```

Loop guard

Recursive type

Invitation Example - Session Type

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer>;
  InfoSvr->Bob: <Location>;
  InfoSvr:
    [InfoSvr->Bob: <Member>;
     InfoSvr->Bob: <Employer>;
     InfoSvr->Bob: <Location>;
     InfoSvr->Bob: <EmailAddr>;
     Bob:
     {INVITE: Bob->MailSvr: <EmailAddr,Event>,
      NOOP:}
    ]*
}
```

Invitation Example - Session Type

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer>;
  InfoSvr->Bob: <Location>;
  InfoSvr:
    [InfoSvr->Bob: <Member>;
     InfoSvr->Bob: <Employer>;
     InfoSvr->Bob: <Location>;
     InfoSvr->Bob: <EmailAddr>;
     Bob:
     {INVITE: Bob->MailSvr: <EmailAddr,Event>,
      NOOP:}
    ]*
}
```

Label selection

Invitation Example - Session Type

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer>;
  InfoSvr->Bob: <Location>;
  InfoSvr:
    [InfoSvr->Bob: <Member>;
     InfoSvr->Bob: <Employer>;
     InfoSvr->Bob: <Location>;
     InfoSvr->Bob: <EmailAddr>;
     Bob:
     {INVITE: Bob->MailSvr: <EmailAddr,Event>,
      NOOP:}
    ]*
}
```

Choice guard

Label selection

Invitation Example - Type Driven Optimizations

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer>;
  InfoSvr->Bob: <Location>;
  InfoSvr:
    [InfoSvr->Bob: <Member>;
     InfoSvr->Bob: <Employer>;
     InfoSvr->Bob: <Location>;
     InfoSvr->Bob: <EmailAddr>;
     Bob:
     {INVITE: Bob->MailSvr: <EmailAddr, Event>,
      NOOP:}
    ]*
}
```

Invitation Example - Type Driven Optimizations

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer>;
  InfoSvr->Bob: <Location>;
  InfoSvr:
    [InfoSvr->Bob: <Member>;
     InfoSvr->Bob: <Employer>;
     InfoSvr->Bob: <Location>;
     InfoSvr->Bob: <EmailAddr>;
     Bob:
     {INVITE: Bob->MailSvr: <EmailAddr, Event>,
      NOOP:}
    ]*
}
```

Invitation Example - Type Driven Optimizations

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer>;
  InfoSvr->Bob: <Location>;
  InfoSvr:
     [InfoSvr->Bob: <Member>;
      InfoSvr->Bob: <Employer>;
      InfoSvr->Bob: <Location>;
      InfoSvr->Bob: <EmailAddr>;
      Bob:
      {INVITE: Bob->MailSvr: <EmailAddr, Event>,
       NOOP:}
     ]*
}
```

Multiple contiguous sends can be batched

Invitation Example - Type Driven Optimizations

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer,Location>;
  InfoSvr:
    [InfoSvr->Bob: <Member,Employer, Location, EmailAddr>;
     Bob:
     {INVITE: Bob->MailSvr: <EmailId, Event>,
      NOOP:}
    ]*
}
```

Invitation Example - Type Driven Optimizations

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer,Location>;
  InfoSvr:
    [InfoSvr->Bob: <Member,Employer, Location, EmailAddr>;
     Bob:
     {INVITE: Bob->MailSvr: <EmailId, Event>,
      NOOP:}
    ]*
}
```

Invitation Example - Type Driven Optimizations

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer,Location>;
  InfoSvr:
    [InfoSvr->Bob: <Member,Employer, Location, EmailAddr>;
     Bob:
     {INVITE: Bob->MailSvr: <EmailId, Event>,
      NOOP:}
    ]*
}
```

Can we batch together this recursive type?

Invitation Example - Type Driven Optimizations

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer,Location>;
  InfoSvr:
    [InfoSvr->Bob: <Member,Employer, Location, EmailAddr>;
     Bob:
     {INVITE: Bob->MailSvr: <EmailId, Event>,
      NOOP:}
    ]*
}
```

Can we batch together this recursive type?

No intervening receives by `InfoSvr` in recursive type

Invitation Example - Type Driven Optimizations

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer,Location>;
  InfoSvr->Bob: <Member,Employer, Location, EmailAddr>*;
  Bob:{INVITE: Bob->MailSvr: <EmailId, Event>, NOOP:}*
}
```

Invitation Example - Type Driven Optimizations

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer,Location>;
  InfoSvr->Bob: <Member,Employer, Location, EmailAddr>*;
  Bob:{INVITE: Bob->MailSvr: <EmailId, Event>, NOOP:}*
}
```

- Recursive type unrolling factor is a tunable parameter

Invitation Example - Type Driven Optimizations

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer,Location>;
  InfoSvr->Bob: <Member,Employer, Location, EmailAddr>*;
  Bob:{INVITE: Bob->MailSvr: <EmailId, Event>, NOOP:}*
}
```

- Recursive type unrolling factor is a tunable parameter
- Runtime handles marshaling and unmarshaling the batches

Invitation Example - Exporting Continuations

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer,Location>;
  InfoSvr->Bob: <Member,Employer,Location,EmailAddr>*;
  Bob:{INVITE: Bob->MailSvr: <EmailAddr,Event>, NOOP:}*
}
```

Can we bypass Bob?

Invitation Example - Exporting Continuations

```
protocol invitation {
   participants Bob, InfoSvr, MailSvr;
   Bob: begin;
   InfoSvr->Bob: <Employer,Location>;
   InfoSvr->Bob: <Member,Employer,Location,EmailAddr>*;
   Bob:{INVITE: Bob->MailSvr: <EmailAddr,Event>, NOOP:}*
}
```

Can we bypass Bob?

- Rewriting communication requests

Invitation Example - Exporting Continuations

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer,Location>;
  InfoSvr->Bob: <Member,Employer,Location,EmailAddr>*;
  Bob:{INVITE: Bob->MailSvr: <EmailAddr,Event>, NOOP:}*
}
```

? *Can we bypass Bob?*

- Rewriting communication requests
- Cannot be exported if

Invitation Example - Exporting Continuations

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer,Location>;
  InfoSvr->Bob: <Member,Employer,Location,EmailAddr>*;
  Bob:{INVITE: Bob->MailSvr: <EmailAddr,Event>, NOOP:}*
}
```

? *Can we bypass Bob?*

- Rewriting communication requests
- Cannot be exported if
  - local state is accessed - file, database, system status, etc.,

Invitation Example - Exporting Continuations

```
protocol invitation {
  participants Bob, InfoSvr, MailSvr;
  Bob: begin;
  InfoSvr->Bob: <Employer,Location>;
  InfoSvr->Bob: <Member,Employer,Location,EmailAddr>*;
  Bob:{INVITE: Bob->MailSvr: <EmailAddr,Event>, NOOP:}*
}
```

? *Can we bypass Bob?*

- Rewriting communication requests
- Cannot be exported if
  - local state is accessed - file, database, system status, etc.,
  - system calls are invoked

Invitation Example - Exporting Bob’s Code

```
void invite_coworkers() {
  Event evt = me.createEvent(“party”, date);
  Employer myEmp = me.getEmployer();
  Location myLoc = me.getLocation();
  for (Member friend : me.getFriends()) {
    if (myEmp.equals(friend.getEmployer()) &&
        myLoc.equals(friend.getLocation()) &&
        User.approve(friend)) {
         mailSvr.sendMail(friend.getEmailId(),evt);
    }
  }
```

- Local state/system call ✖

Invitation Example - No Local State Access

```
void invite_coworkers’() {
  Event evt = me.createEvent(“party”, date);
  Employer myEmp = me.getEmployer();
  Location myLoc = me.getLocation();
  for (Member friend : me.getFriends()) {
    if (myEmp.equals(friend.getEmployer()) &&
        myLoc.equals(friend.getLocation())) {
         mailSvr.sendMail(friend.getEmailId(),evt);
    }
  }
```

Invitation Example - No Local State Access

```
void invite_coworkers’() {
  Event evt = me.createEvent(“party”, date);
  Employer myEmp = me.getEmployer();
  Location myLoc = me.getLocation();
  for (Member friend : me.getFriends()) {
    if (myEmp.equals(friend.getEmployer()) &&
        myLoc.equals(friend.getLocation())) {
         mailSvr.sendMail(friend.getEmailId(),evt);
    }
  }
```

- Executed at `InfoSvr`

Invitation Example - No Local State Access

```
void invite_coworkers’() {
  Event evt = me.createEvent(“party”, date);
  Employer myEmp = me.getEmployer();
  Location myLoc = me.getLocation();
  for (Member friend : me.getFriends()) {
    if (myEmp.equals(friend.getEmployer()) &&
        myLoc.equals(friend.getLocation())) {
         mailSvr.sendMail(friend.getEmailId(),evt);
    }
  }
```

- Executed at `InfoSvr`
- `me` and `friend` are local objects

Invitation Example - No Local State Access

```
void invite_coworkers’() {
  Event evt = me.createEvent(“party”, date);
  Employer myEmp = me.getEmployer();
  Location myLoc = me.getLocation();
  for (Member friend : me.getFriends()) {
    if (myEmp.equals(friend.getEmployer()) &&
        myLoc.equals(friend.getLocation())) {
         mailSvr.sendMail(friend.getEmailId(),evt);
    }
  }
```

- Executed at `InfoSvr`
- `me` and `friend` are local objects
- Only remote operation is `sendMail()`, which is also batched

Experimental Setup

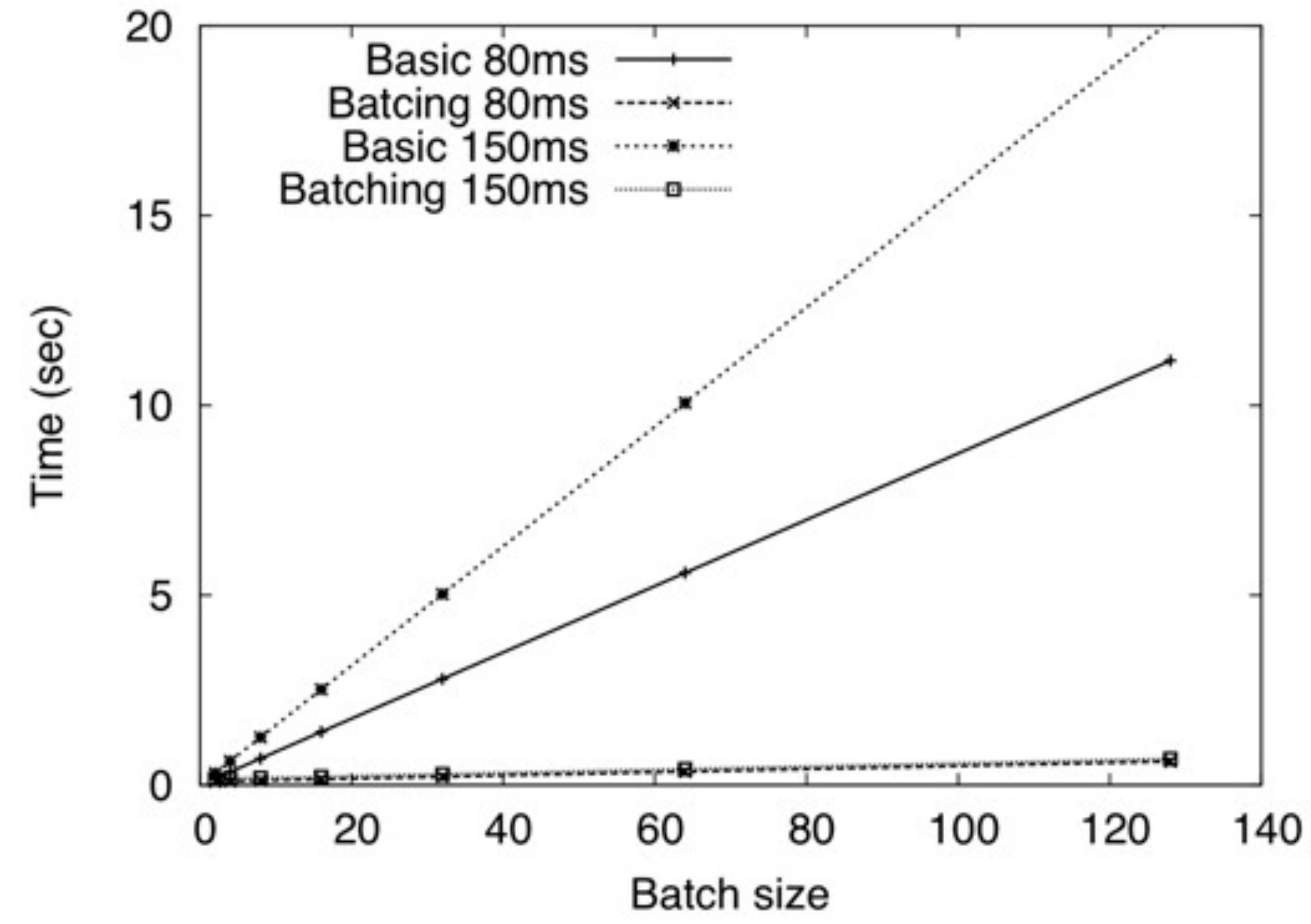
- Benchmarks
  - Batching
  - Exporting continuations
- Batching experiments were conducted on Emulab
- Emulab machines were 850 MHz Intel Pentium 3 with 512 MB of RAM

Batching

- 2 Emulab nodes with 1MBPS link.

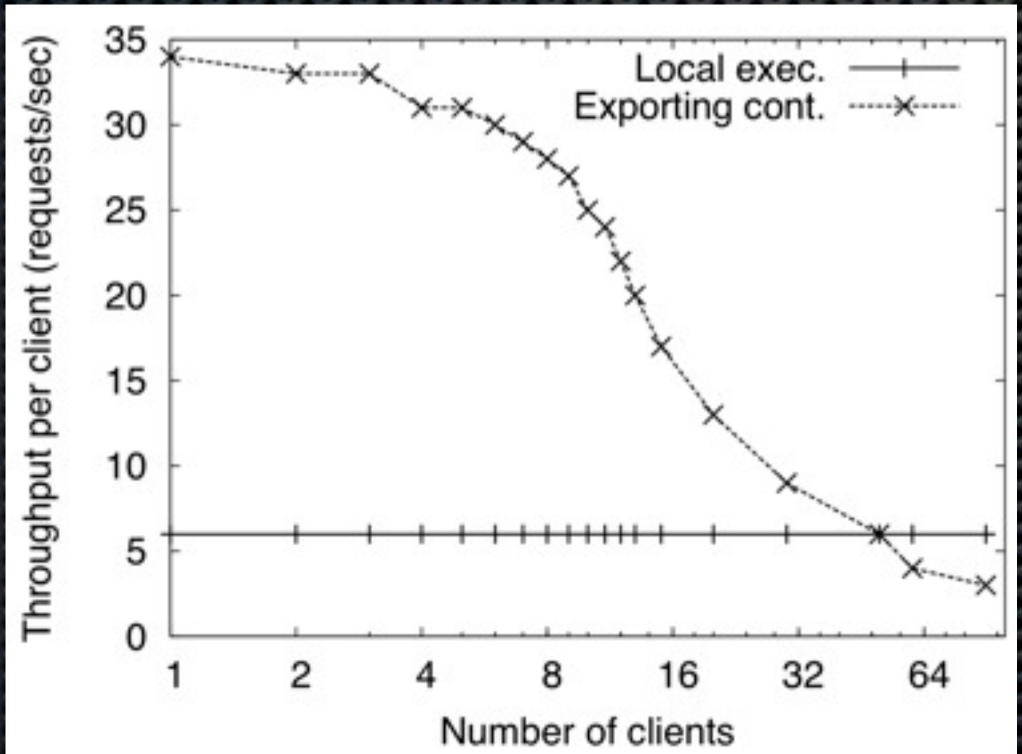
```
client:
[client->server: <Signature>;
 server->client: <bool>]*
```

- Tested for various RTT and signature sizes
- Batching performs well and the overhead is very little

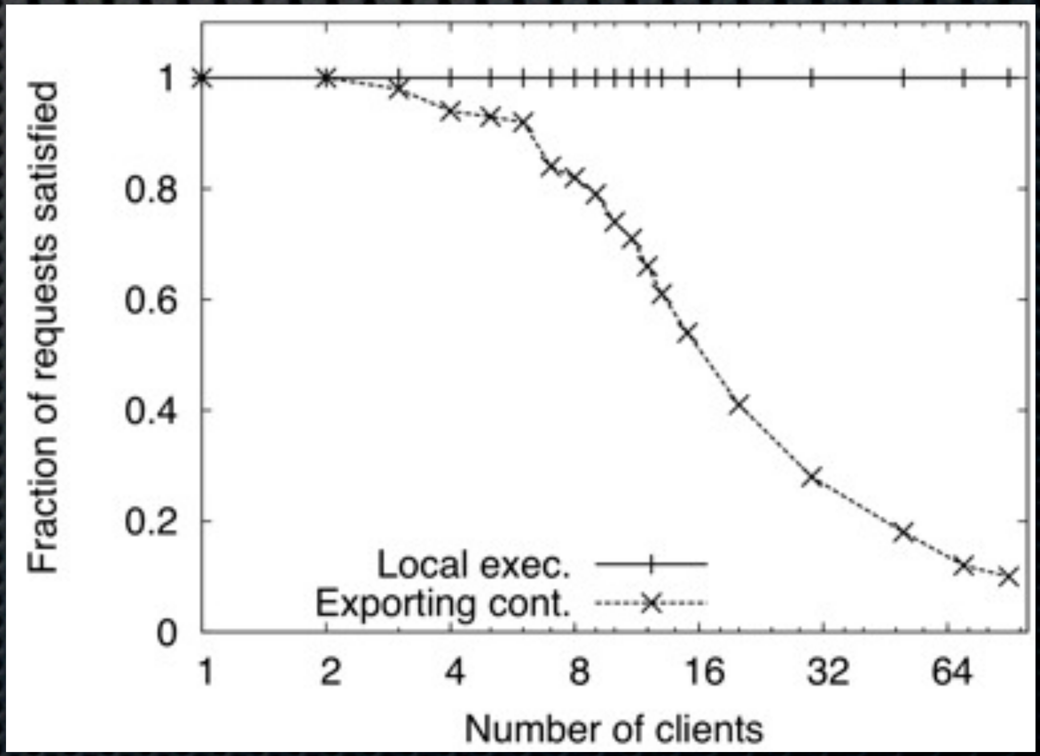


Exporting Continuation

- Algorithmic trading
  - Remote methods - `fetchQuotes()` and `doTrading()`
  - Local/exported method - `findTradingOptions()`
- Server configuration - dual core machine - 3 GHz and 4GB RAM
- Client configuration - Intel Pentium II 500 MHz



client throughput



server throughput

Conclusion

- Limitations
  - Aggressive continuation exporting can overload participants
  - Security issues with client code executing on the server
- Future Work
  - User annotations for continuation exporting
  - Group communication abstraction
  - Formally prove that the transformations are correct

Questions?

Extra slides - Session Implementation

```
protocol simple@Bob{
   !begin;
   DeepThought:!<string>;
   DeepThought:?<int>;
}
```

Extra slides - Session Implementation

```
SessionRegistry.instantiate(simple,“session1”);
```

```
protocol simple@Bob{
  !begin;
  DeepThought:!<string>;
  DeepThought:?<int>;
}
```

Extra slides - Session Implementation

```
protocol simple@Bob{
   !begin;
   DeepThought:!<string>;
   DeepThought:?<int>;
}
```

```
SessionRegistry.instantiate(simple,“session1”);
SessionSocket ss =
   SessionRegistry.lookup(simple,“session1”,Bob);
```

Extra slides - Session Implementation

```
protocol simple@Bob{
  !begin;
  DeepThought:!<string>;
  DeepThought:?<int>;
}
```

```
SessionRegistry.instantiate(simple,“session1”);
SessionSocket ss =
   SessionRegistry.lookup(simple,“session1”,Bob);
ss.begin ();
```

Extra slides - Session Implementation

```
protocol simple@Bob{
  !begin;
  DeepThought:!<string>;
  DeepThought:?<int>;
}
```

```
SessionRegistry.instantiate(simple,“session1”);
SessionSocket ss =
   SessionRegistry.lookup(simple,“session1”,Bob);
ss.begin ();
ss.send (DeepThought, “what is the ultimate
   answer to life, universe, and everything?”);
```

Extra slides - Session Implementation

```
protocol simple@Bob{
  !begin;
  DeepThought:!<string>;
  DeepThought:?<int>;
}
```

```
SessionRegistry.instantiate(simple,“session1”);
SessionSocket ss =
   SessionRegistry.lookup(simple,“session1”,Bob);
ss.begin ();
ss.send (DeepThought, “what is the ultimate
   answer to life, universe, and everything?”);
int answer = ss.receive (DeepThought);
```