Flux

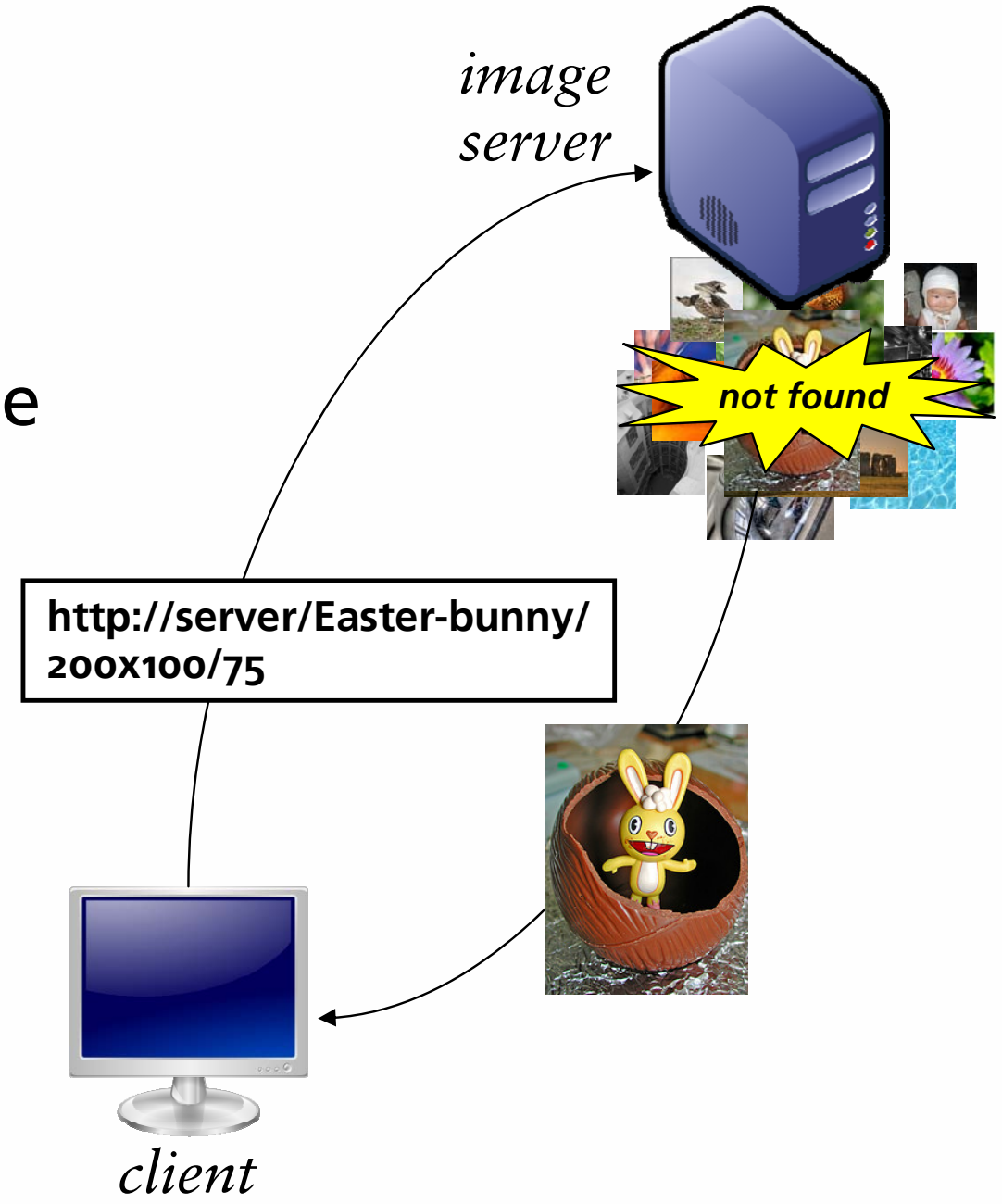
A Language for Programming High-Performance Servers

Brendan Burns, Kevin Grimaldi, Alex Kostadinov, Emery Berger, Mark Corner

University of Massachusetts Amherst

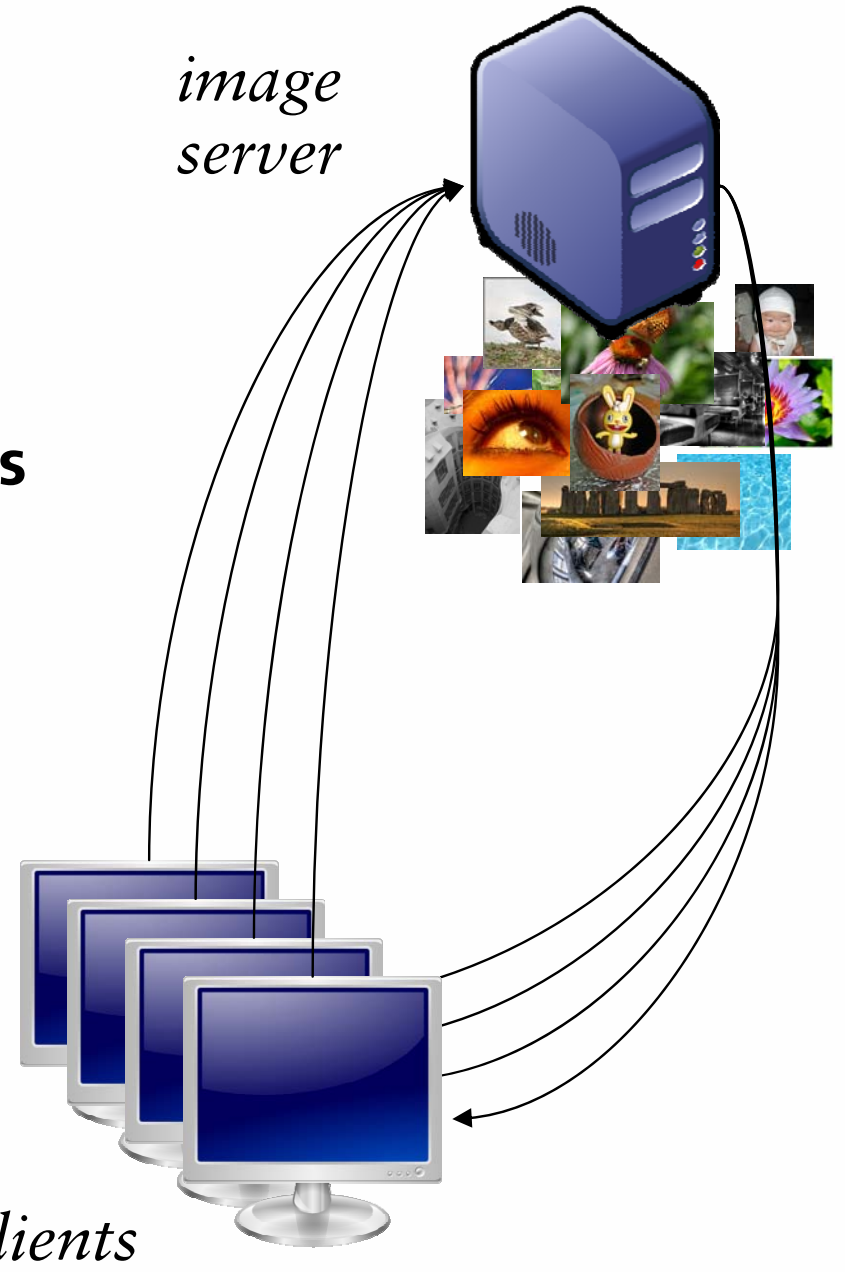
Motivation: Image Server

- **Client**
  - Requests image @ desired quality, size
- **Server**
  - Images: RAW
  - Compresses to JPG
  - Caches requests
  - Sends to client



Problem: Concurrency

- Sequential fine until:
  - More **clients**
  - Bigger **server**
    - **Multicores, multiprocessors**
- One approach: **threads**
  - **Limit reuse, risk deadlock, burden programmer**
  - Complicate debugging
  - Mixes program logic & concurrency control



The Flux Programming Language

High-performance & deadlock-free
concurrent programming w/ sequential components

- **Flux = Components + Flow + Atomicity**
  - Components = off-the-shelf C, C++, or Java
  - Flow = path through components
    - **Implicitly parallel**
  - Atomicity = lightweight constraints

- Compiler generates:
  - ***Deadlock-free* server**
    - Runtime independent (threads, events, …)
  - **Discrete event simulator**

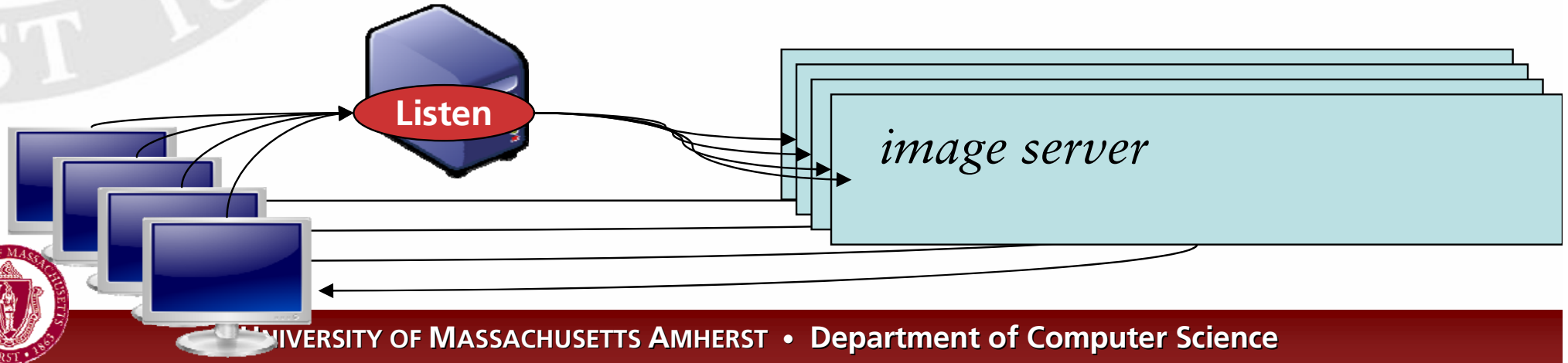
Outline

- **Intro to Flux: building a server**
  - Components
  - Flows
  - Atomicity
- Performance results
  - Server performance
  - Performance prediction
- Future work

Flux Server: Main

- **Source**: one flow per connection
  - Conceptually: separate thread
  - Executes inside implicit **infinite loop**

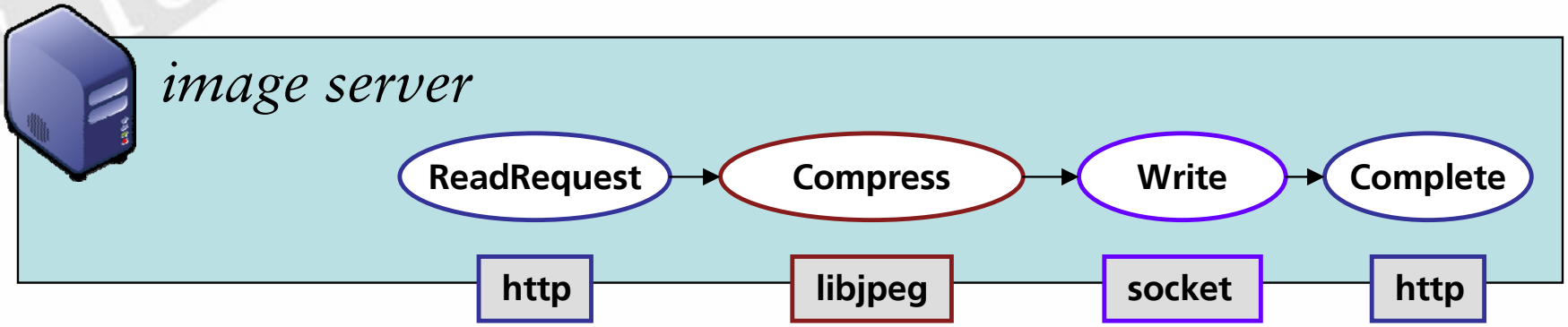
```
source Listen → Image;
```



Flux Image Server

- Basic image server requires:
  - HTTP parsing (**http**)
  - Socket handling (**socket**)
  - JPEG compression (**libjpeg**)
- Single flow implements basic server:

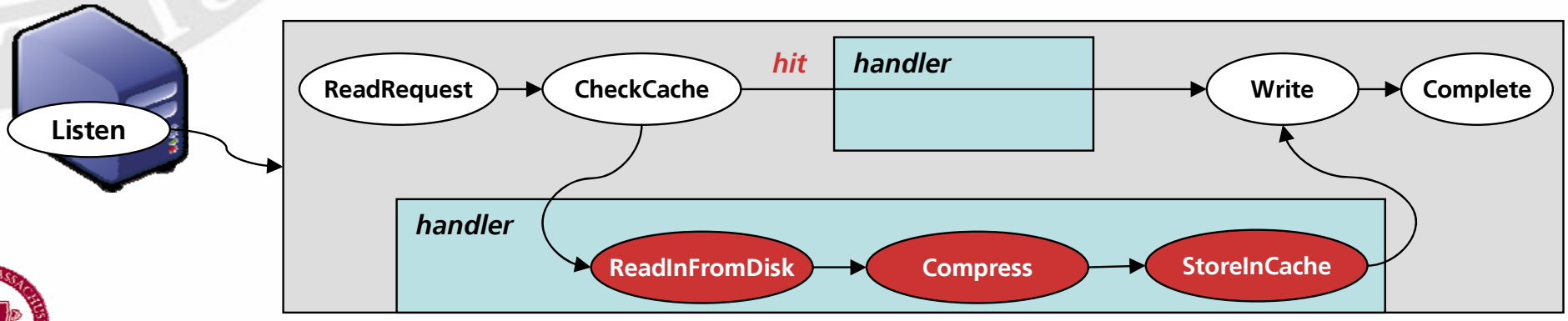
```
Image = ReadRequest → Compress → Write → Complete;
```



Adding Caching

- Cache frequently requested images
  - Increase performance
- Direct data flow with *predicates*
  - Type test applied to output

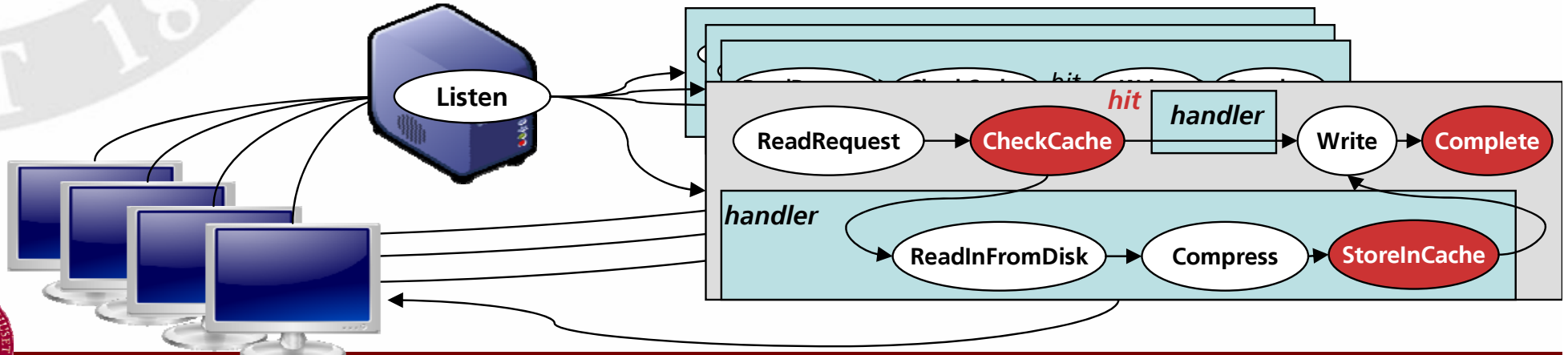
```
typedef hit TestInCache;
Handler:[_,_,hit] = ;
Handler:[_,_,_] = ReadFromDisk → Compress → StoreInCache;
```



Supporting Concurrency

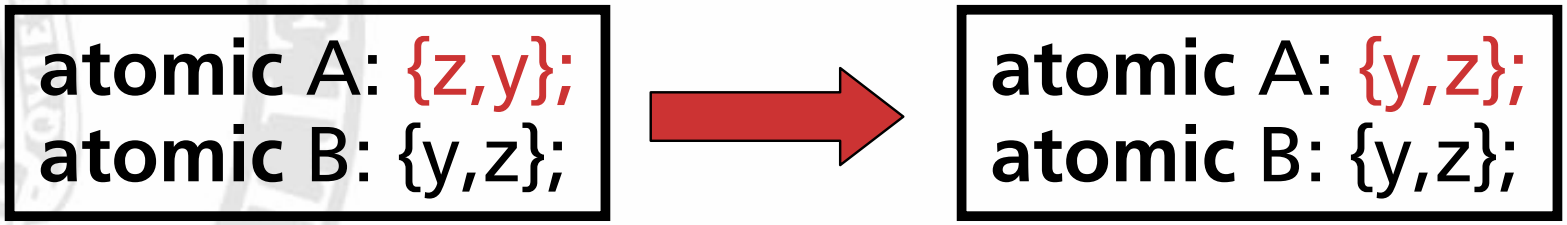
- Many clients = concurrent flows
  - Must keep cache consistent
- *Atomicity constraints*
  - Same name = mutual exclusion
  - Multiple names, reader/writer, per-client (see paper)

```
atomic CheckCache      {cacheLock};
atomic Complete                {cacheLock};
atomic StoreInCache    {cacheLock};
```



Preventing Deadlock

- Naïve execution can deadlock

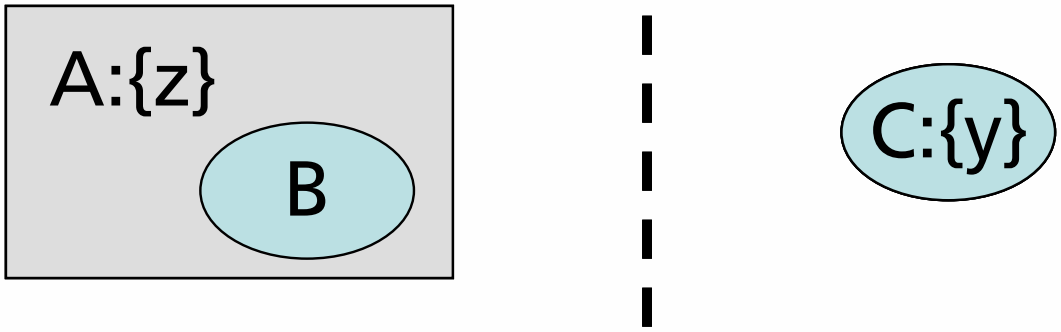


- Establish **canonical lock order**
  - Currently alphabetic by name

Preventing Deadlock, II

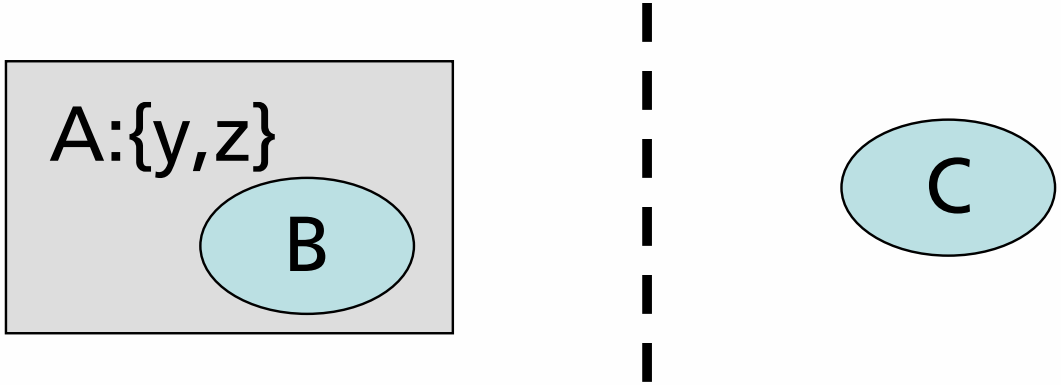
- Harder with abstract nodes

```
A = B;
C = D;
atomic A{z};
atomic B{y};
atomic C{y,z}
```



- Solution: **Elevate** constraints

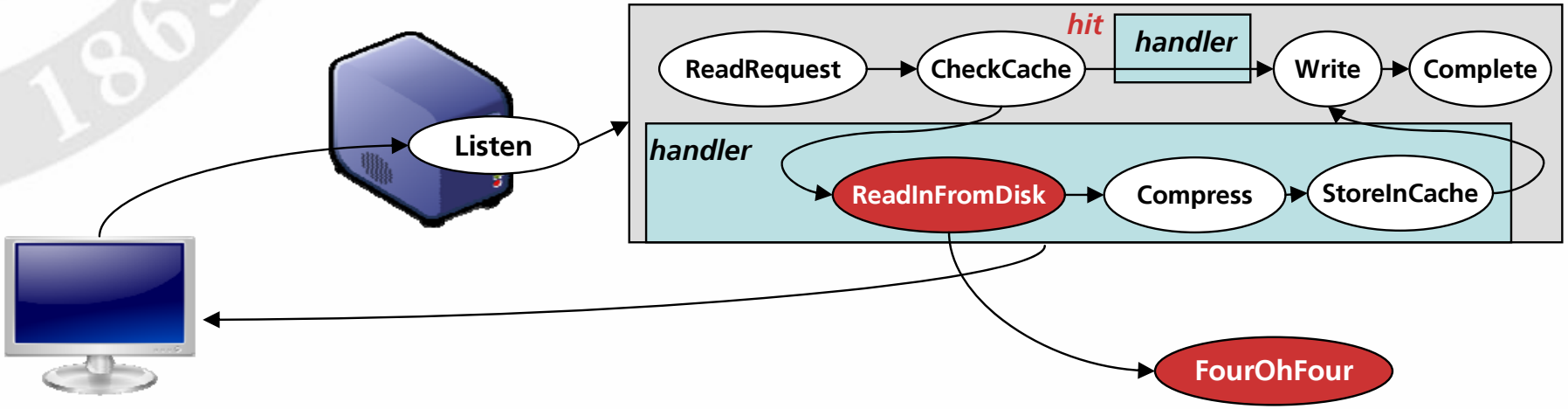
```
A = B;
C = D;
atomic A{y,z};
atomic B{y};
atomic C{y,z}
```



Finally: Handling Errors

- What if image requested doesn't exist?
  - Error = negative return value from component
- Can designate *error handlers*
  - Go to alternate paths on error

handle error ReadInFromDisk → FourOhFour;



Complete Flux Image Server

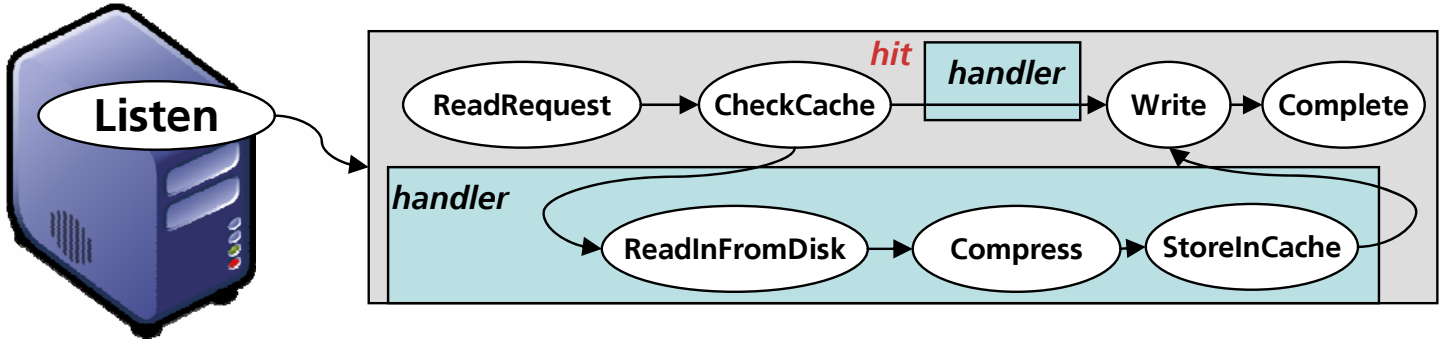
```
source Listen → Image;
Image =
  ReadRequest → CheckCache → Handler → Write → Complete;
Handler[_,_,hit] = ;
Handler[_,_,_] = ReadFromDisk → Compress → StoreInCache;

atomic CheckCache:      {cacheLock};
atomic StoreInCache:    {cacheLock};
atomic Complete:                    {cacheLock};

handle error ReadInFromDisk → FourOhFour;
```

- Concise, readable expression of **server** logic
  - No threads, etc.: simplifies programming, debugging

image server



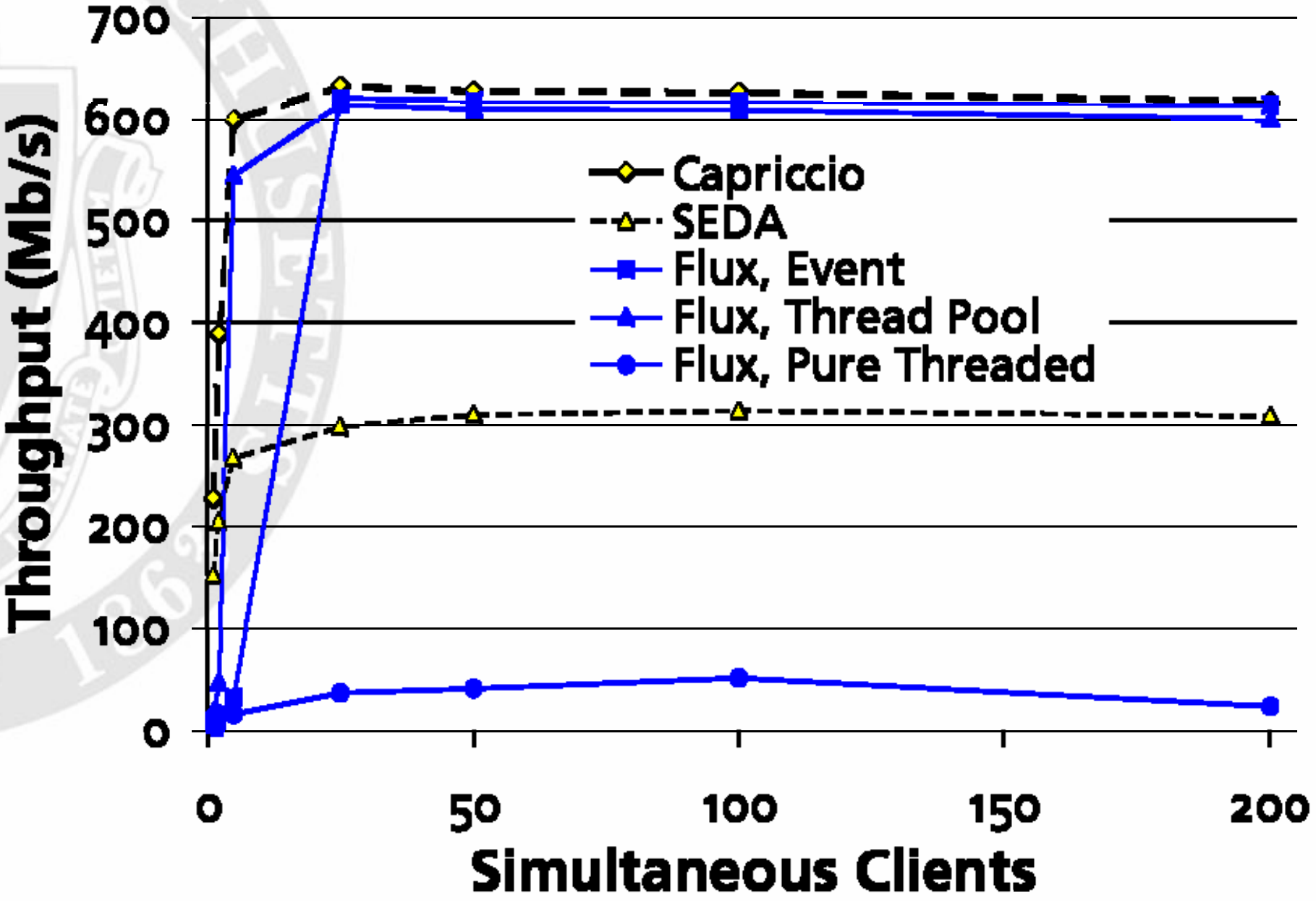
Outline

- Intro to Flux: building a server
  - Components, flow
  - Atomicity, deadlock avoidance
- **Performance results**
  - Server performance
  - Performance prediction
- Future work

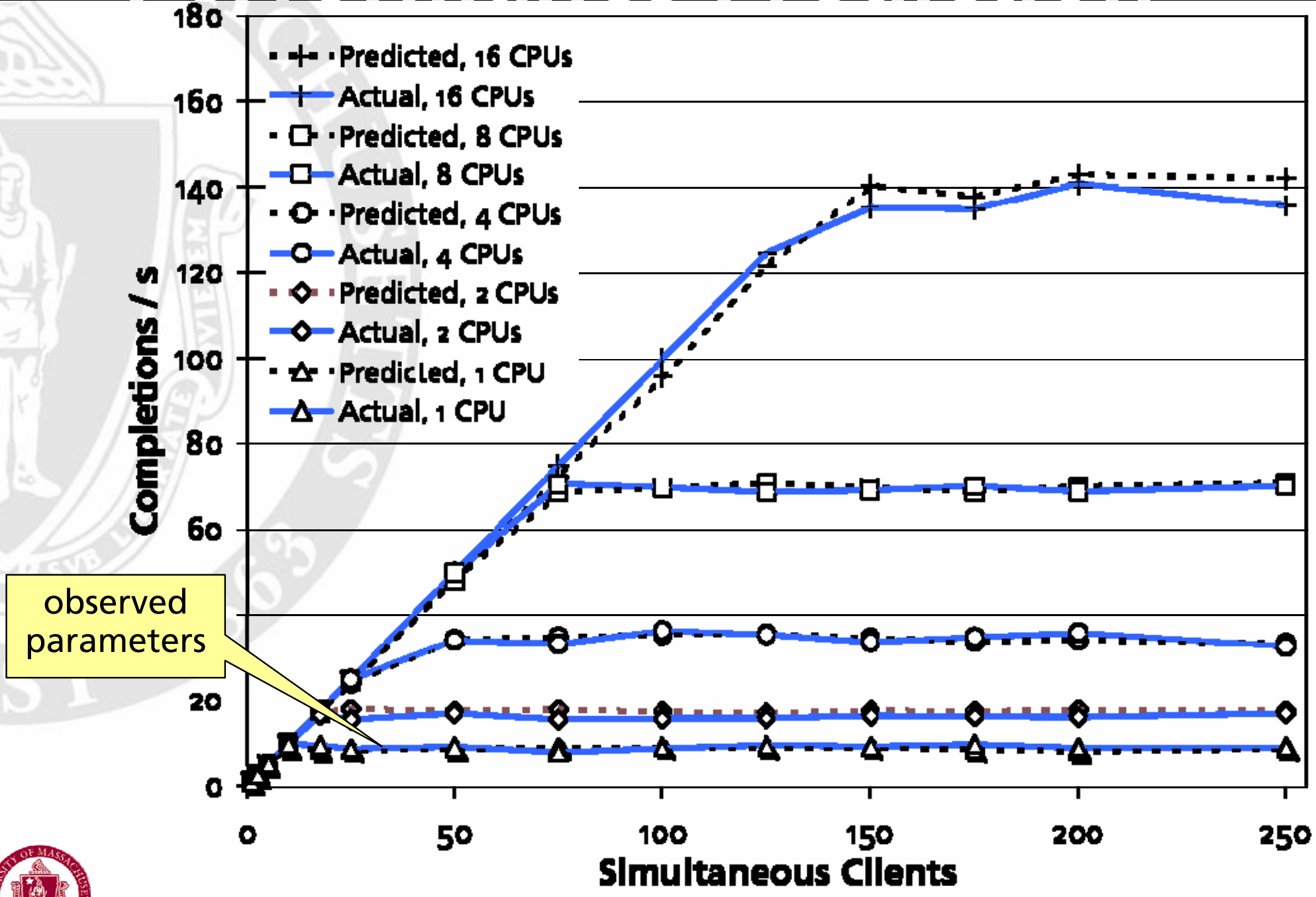
Results

- Four servers:
  - Image server (+ libjpeg) [23 lines of Flux]
  - Multi-player online game [54]
  - BitTorrent (2 undergrads: 1 week!) [84]
  - **Web server** (+ PHP) [36]
- Evaluation
  - Benchmark: variant of **SPECweb99**
  - Three different runtimes
    - Thread, Thread pool, Event-Driven
  - Compared to Capriccio [SOSP03], SEDA [SOSP01]

Web Server


Throughput (Mb/s)
700
600
500
400
300
200
100
0
0
50
100
150
200
Simultaneous Clients
Capriccio
SEDA
Flux, Event
Flux, Thread Pool
Flux, Pure Threaded

Performance Prediction



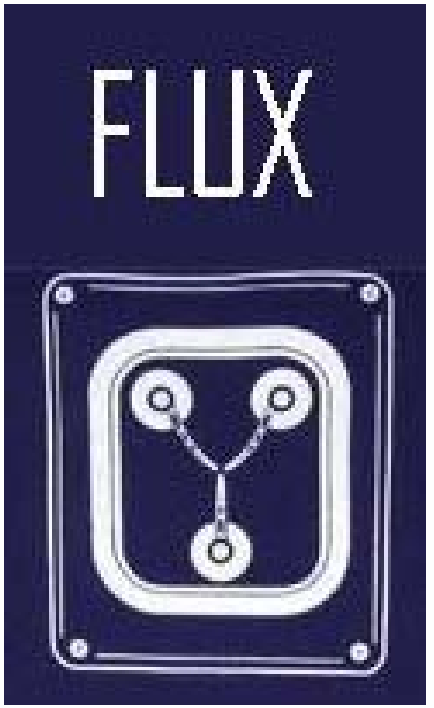
observed parameters

Future Work

- Different runtimes
- Distributed architectures
  - Clusters
- Embedded, power-aware systems
  - Turtles!
  - Embedded space similar to servers
  - eFlux: includes power constraints
    - Removes/adds flows dynamically in response to power

Conclusion

- Flux: language for server programming
  - Uses sequential code
  - Separates logic and runtime
  - Deadlock-free high-performance servers + simulators
- **`flux.cs.umass.edu`**
  - Hosted by Flux web server; download via Flux BitTorrent



flux: from Latin *fluxus*, p.p. of *fluere* = "to flow"

Backup Slides

Relaxed Atomicity

- Reader / writer constraints
  - Multiple readers **or** single writer

```
atomic ReadList:    {listAccess?};
atomic AddToList: {listAccess!};
```

- Per-session constraints
  - One constraint per client / *session*

```
atomic AddHasChunk: {chunks(session)};
```