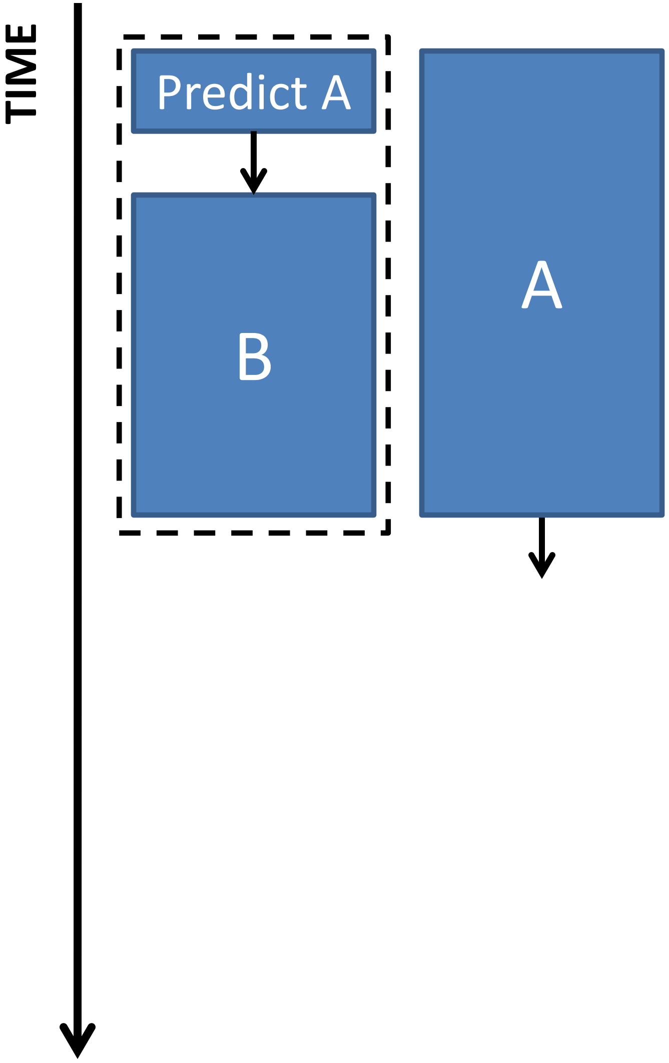
Operating System Support for Application-Specific Speculation

Benjamin Wester

Peter Chen and Jason Flinn

University of Michigan

Speculative Execution



- Sequential dependent tasks
- Predict results of Task A to break dependence
- Execute Task B in parallel
  - Isolate all effects
- Correct prediction: commit
- Wrong prediction: abort

Speculation Everywhere!

- Discrete event simulation
- I/O prefetching
- Distributed shared memory
- Distributed file systems
- Deadlock detection
- Remote displays
- Web page pre-rendering

Speculation as a Service to Apps

How is this system designed?

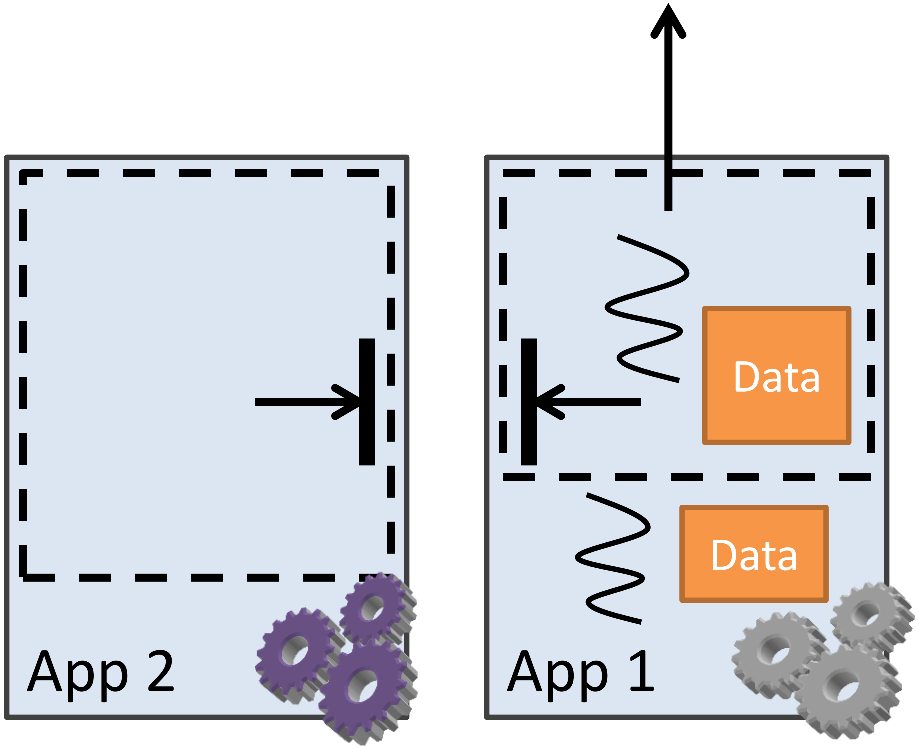
In what ways can it be customized for an app?

How can those customizations be specified?

Outline

- Introduction
- Designing Speculation as a Service
- Implementation
- Evaluation
- Conclusion

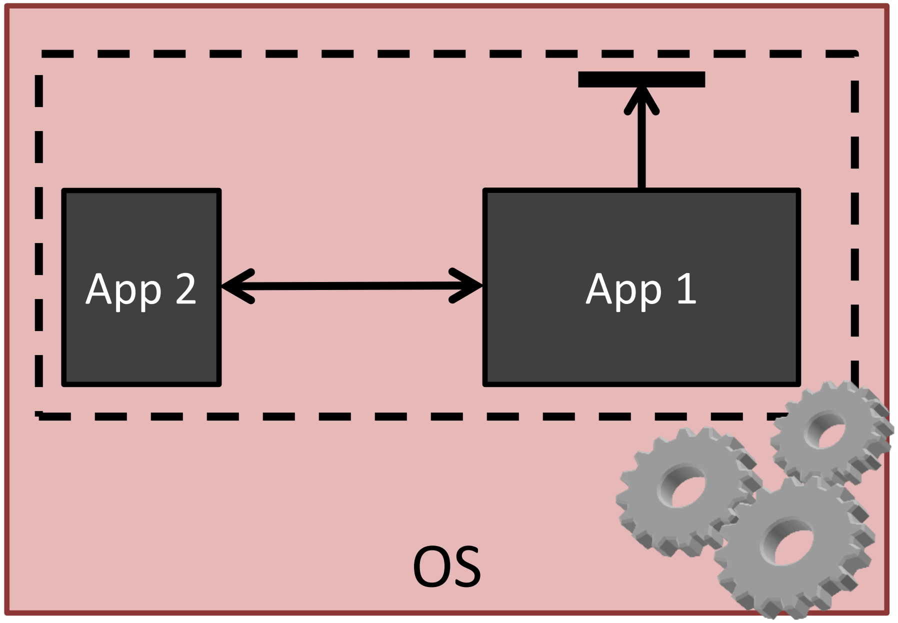
Design 1: In-App Speculation



- + Complete semantic info
- + Predict **arbitrary app operations**
- + Safe operations **allowed**

- – **No reuse**: significant development needed
- – **Scope is limited**: unsafe operations block

Design 2: Generic OS Speculation



- + Apps need **no modifications**
- + **Wide scope**: unsafe operations taint

- – Lacks semantic understanding of app
- – Predict **system calls** only
- – Handle application **conservatively**

Separate Mechanism and Policy

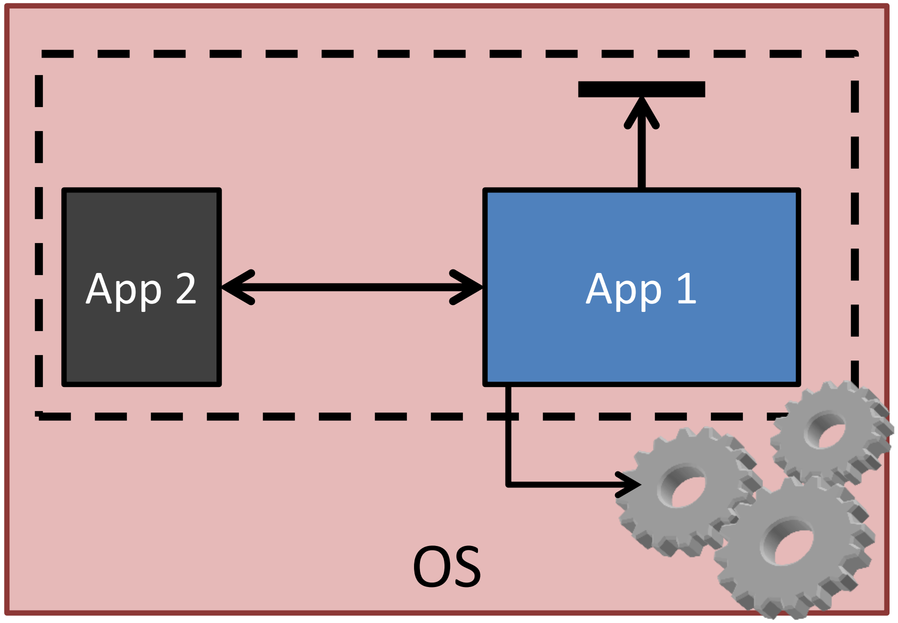
Mechanism implements isolation

Policy describes customizations

Best of both extremes

- Mechanism built in OS
  - Common implementation
  - Wide scope
- Policy specified in Applications
  - Expose semantic information

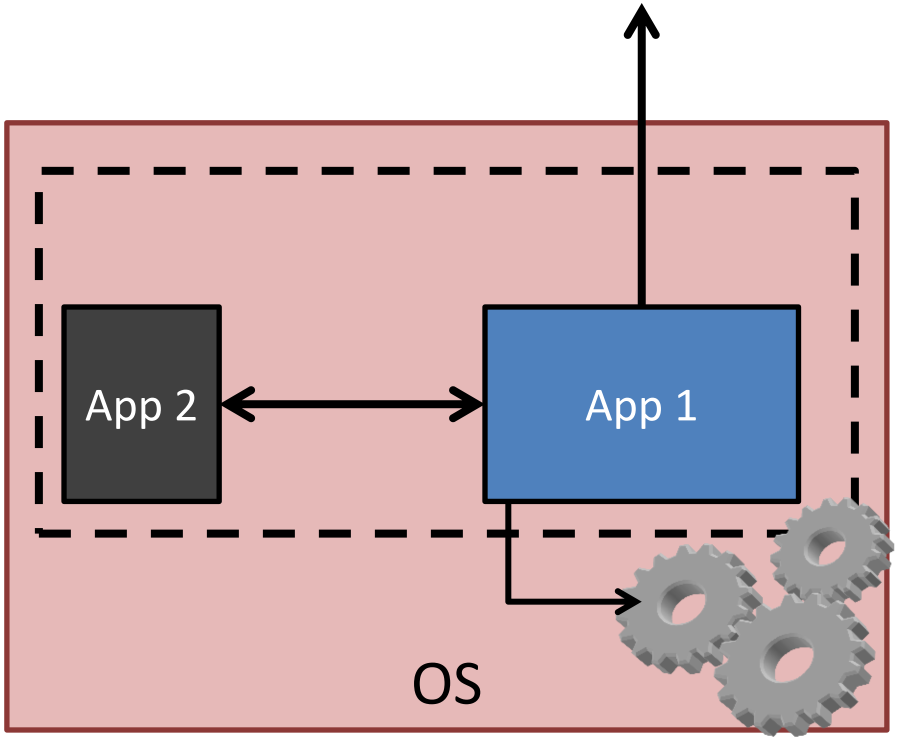
Design 3: Expose Predictions



- + Predict **arbitrary app operations**
- + **Reuse** OS mechanism (with app assistance)
- + **Wide scope** for taint propagation

- – Limited semantic info
  - – Speculative external output **never allowed**
  - – Commit on **identical** results

Design 4: Expose Safety



- + Predict arbitrary app operations
- + Reuse OS mechanism (with app assistance)
- + Wide scope for taint propagation
- + More semantic info
  - + Allow **safe** output
  - + Commit on **equivalent** results

Customizable Policy

- Creation
  - What tasks are predictable
  - How to predict them
- Output
  - What output is safe to allow
- Commit
  - Which results are acceptable to commit

Outline

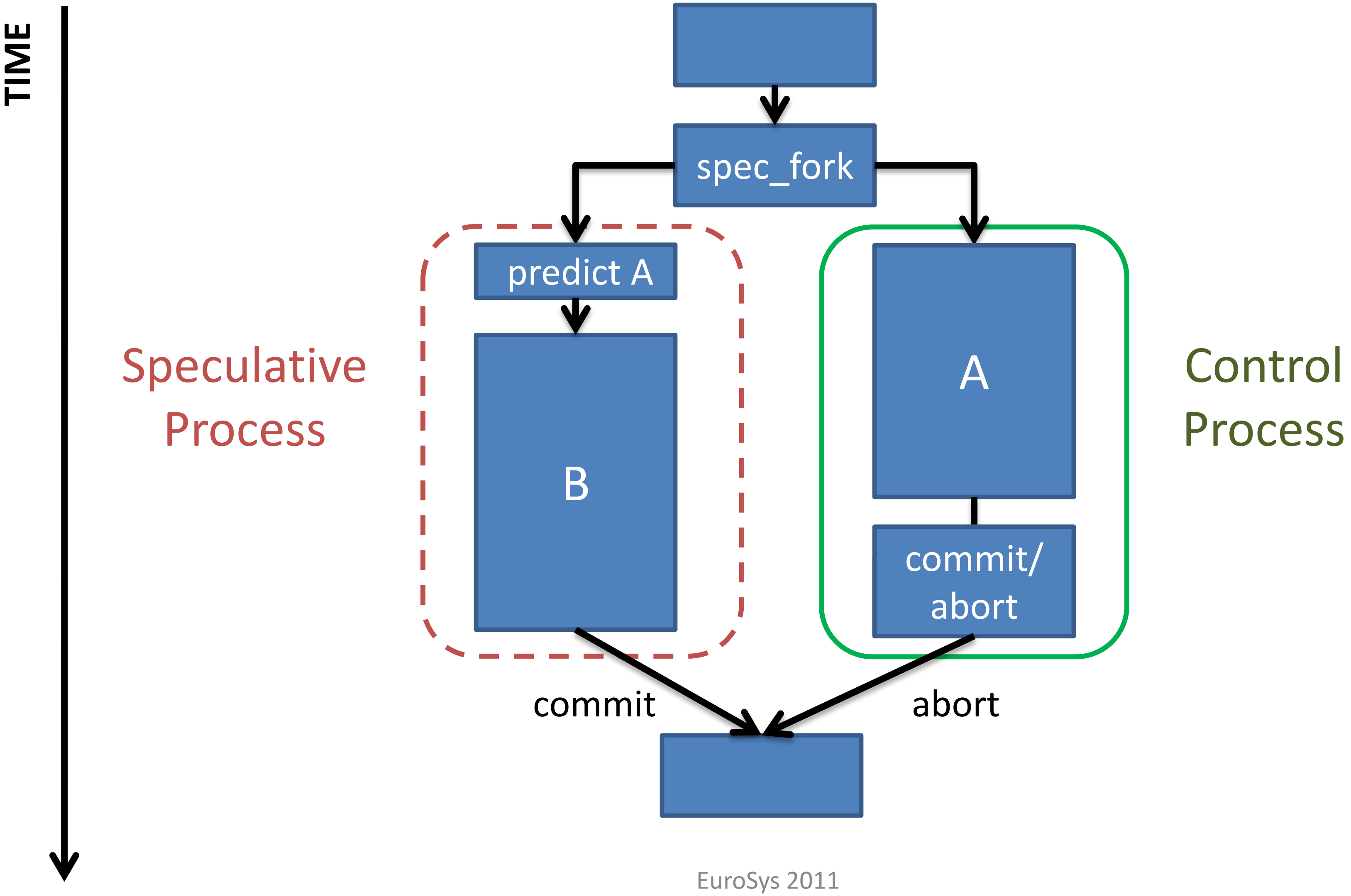
- Introduction
- Designing Speculation as an OS Service
- Implementation
- Evaluation
- Conclusion

Implementation

- Mechanism built in OS
  - Based on Speculator kernel
  - Checkpoints & logs processes, files, IPC, etc.

- Policies expressed using system call API

spec_fork()


TIME
spec_fork
predict A
B
A
commit/ abort
Speculative Process
Control Process
commit
abort

API Example

```
int main() {
        int x;
        int prediction = get_prediction();
        if (spec_fork() == SPECULATIVE) {
                x = prediction;
        } else {
                x = slow_function();
                if (equiv(x, prediction))
                        commit();
                else
                        abort();
        }
        set_output_policy(stdout, ALLOW);
        printf(“%d”, x);
}
```

Creation Policy

Commit Policy

Output Policy

Outline

- Introduction
- Designing Speculation as an OS Service
- Implementation
- Evaluation
- Conclusion

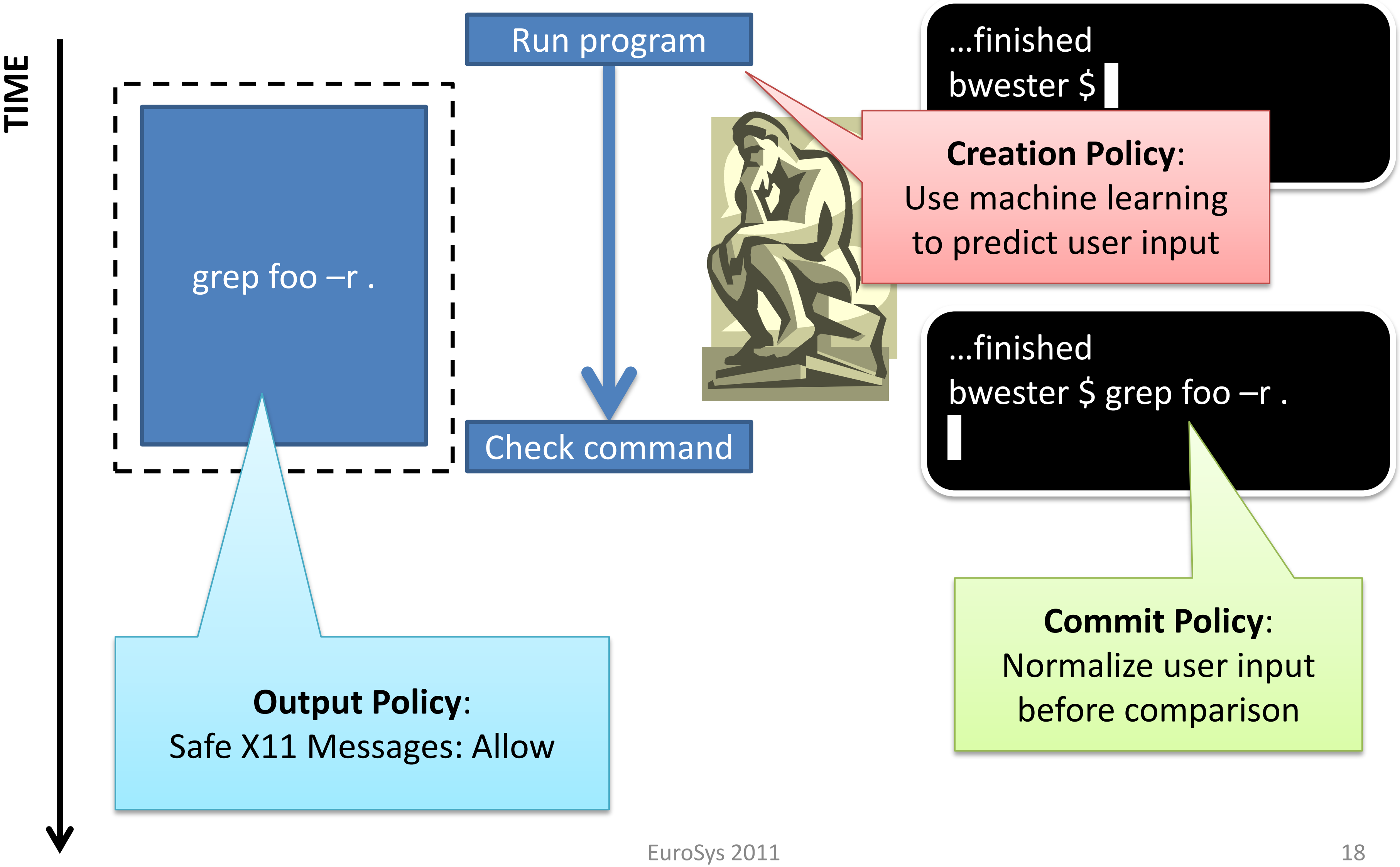
Evaluation

Can apps effectively use API
to increase parallelism?

Case studies

1. Predictive application launching in Bash
2. SSL certificate checks in Firefox
3. Replicated service in PBFT-CS

App 1: Predictive Launching in Bash



```
…finished
bwester $ █
```

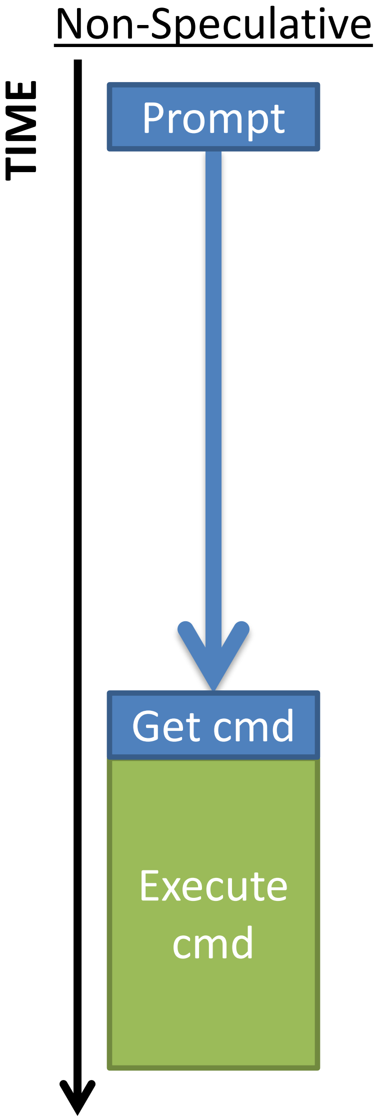
Creation Policy: Use machine learning to predict user input

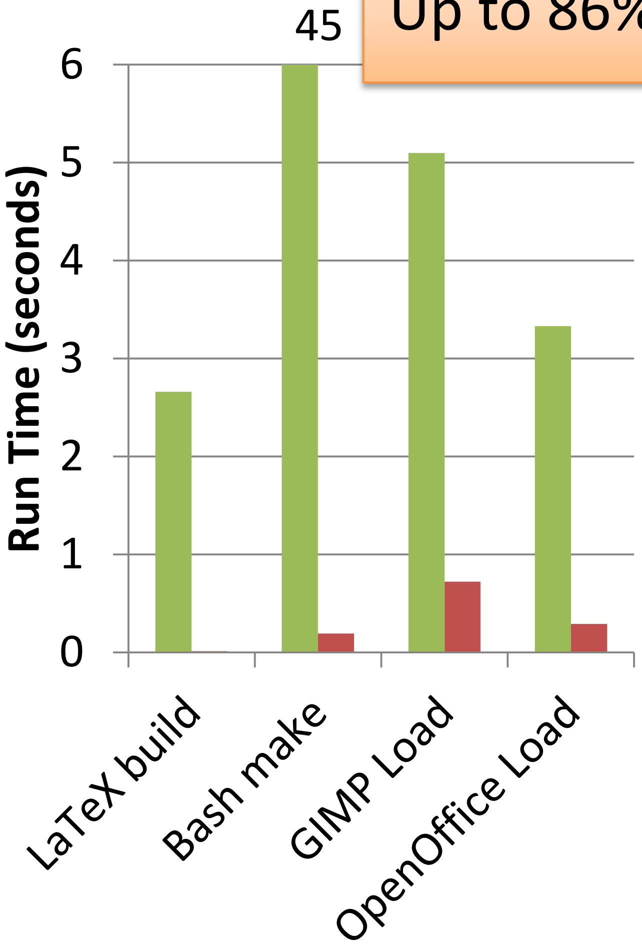
```
…finished
bwester $ grep foo –r .
█
```

Commit Policy: Normalize user input before comparison

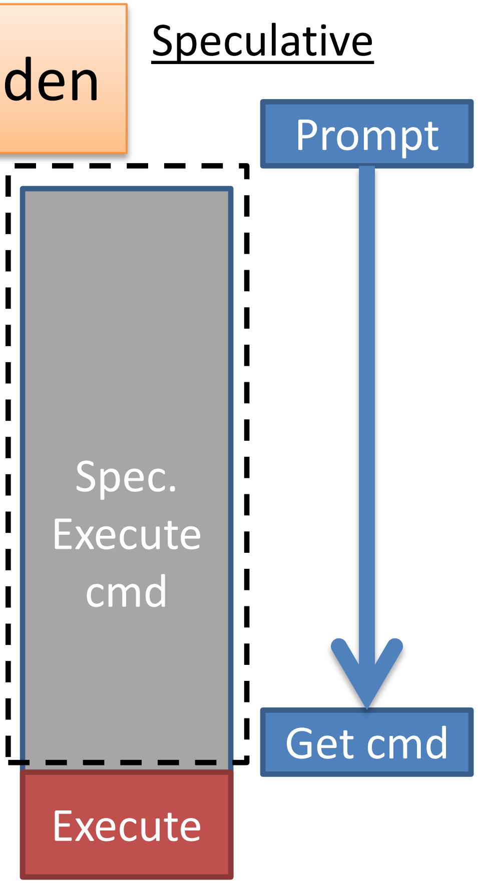
Output Policy: Safe X11 Messages: Allow

How Much Work Can Be Hidden?

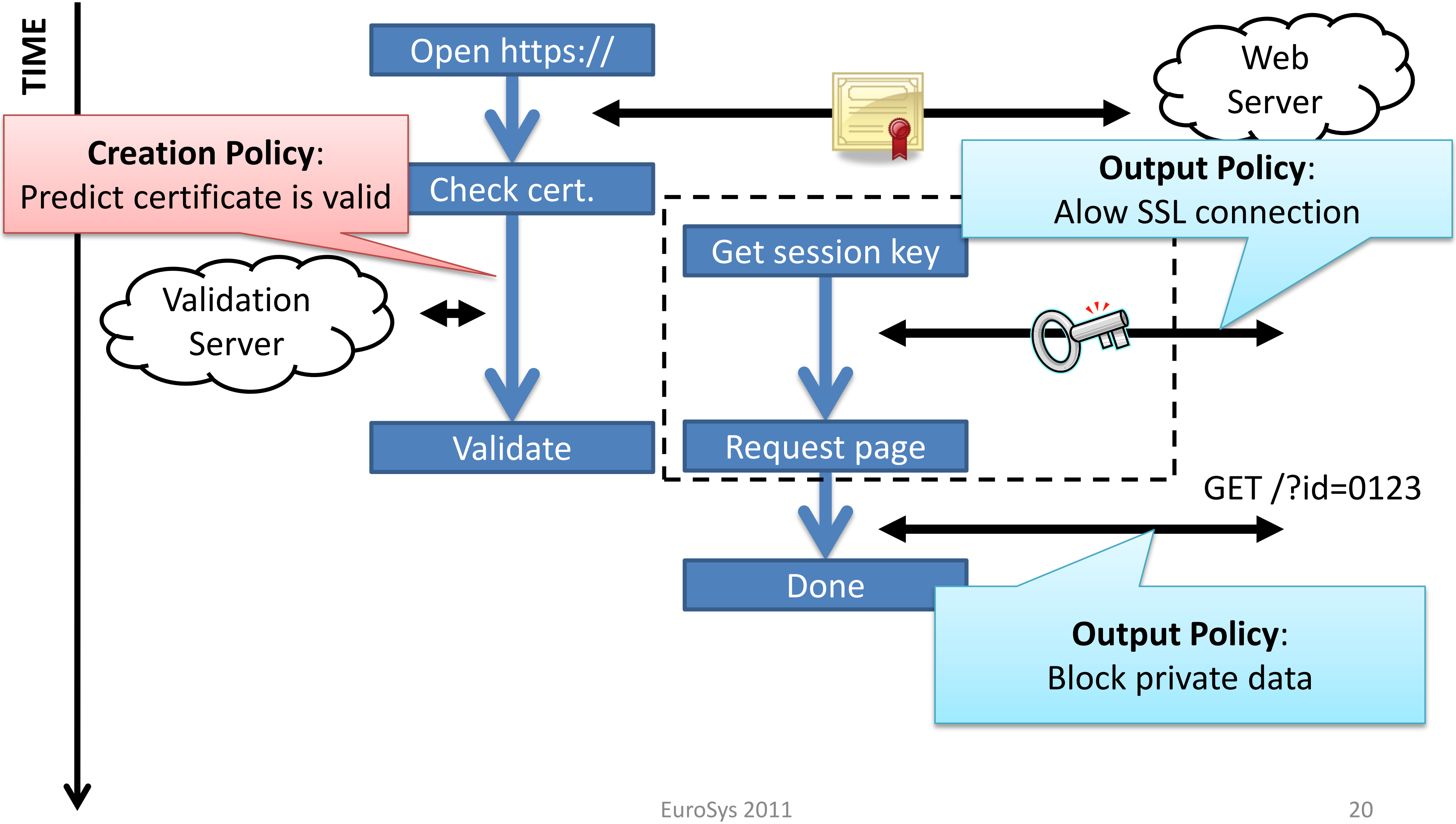




Up to 86% hidden



App 2: Firefox SSL Connections



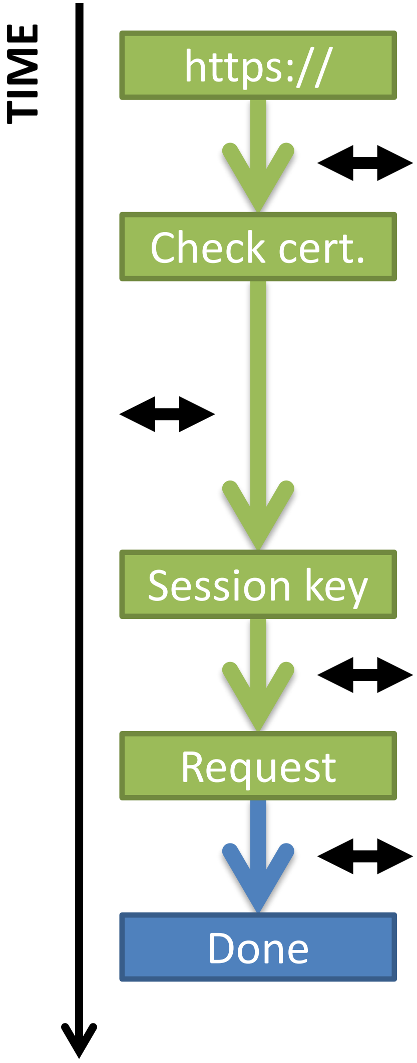
Creation Policy: Predict certificate is valid

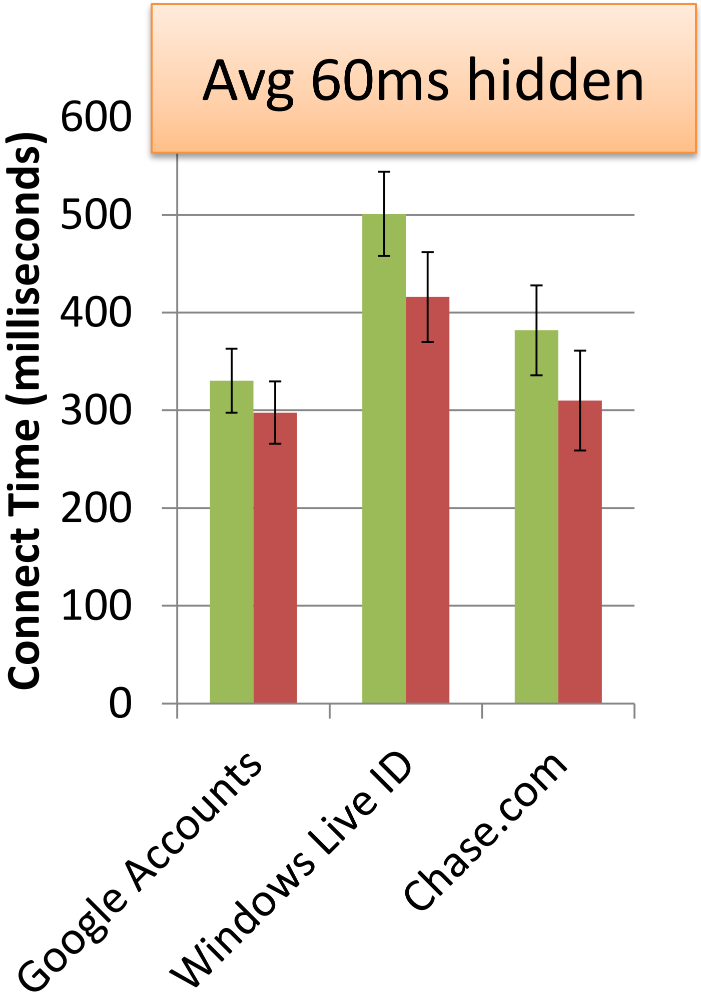
Output Policy: Alow SSL connection

Output Policy: Block private data

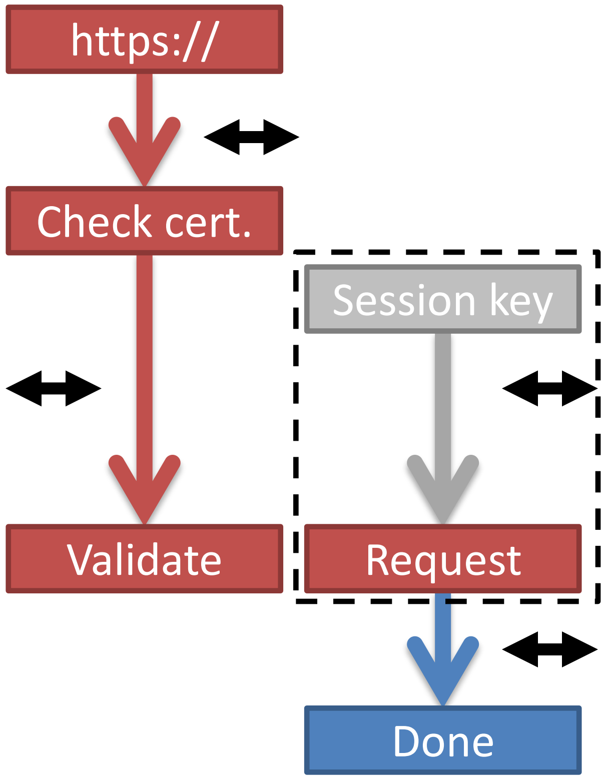
Connection Latency Hidden?

Non-Speculative

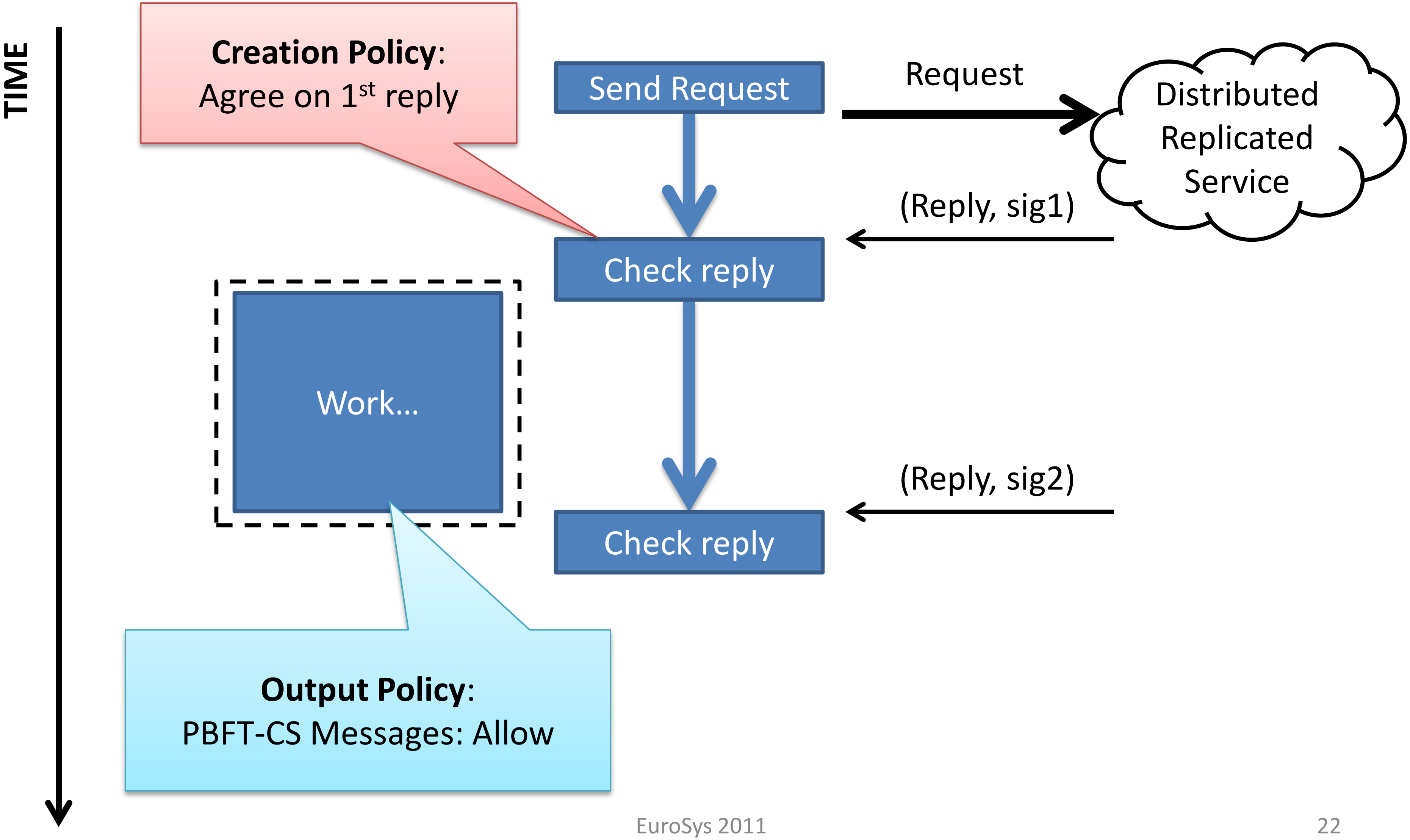



Avg 60ms hidden

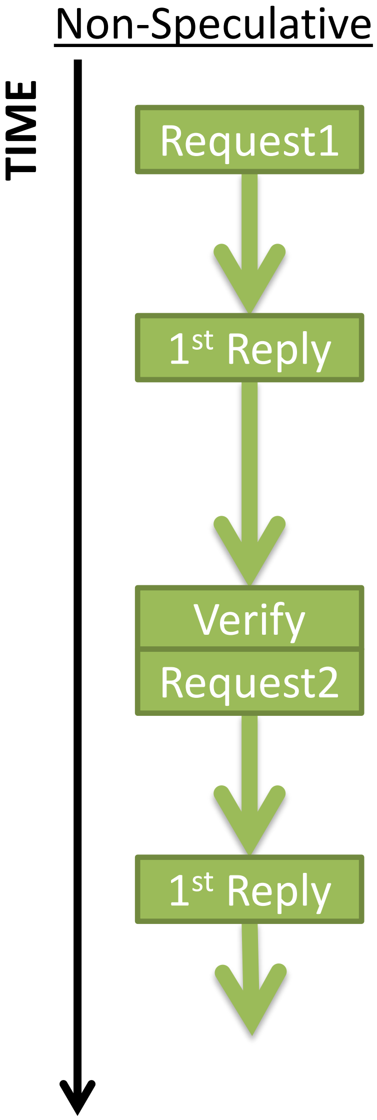
Speculative

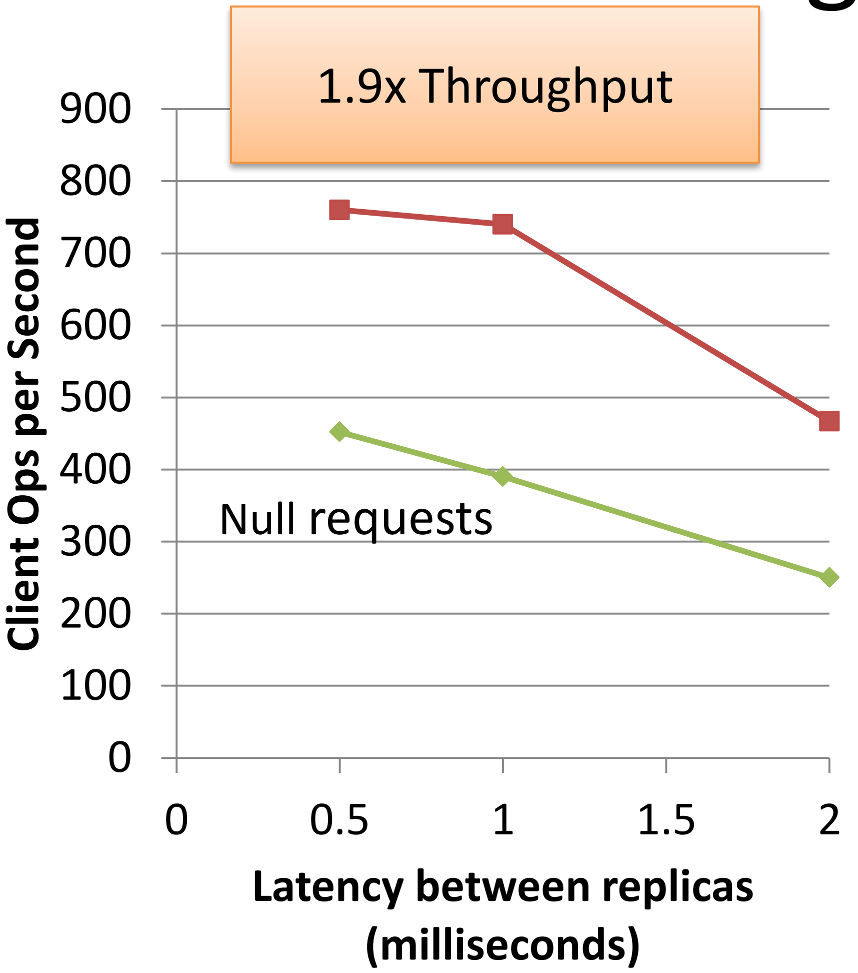


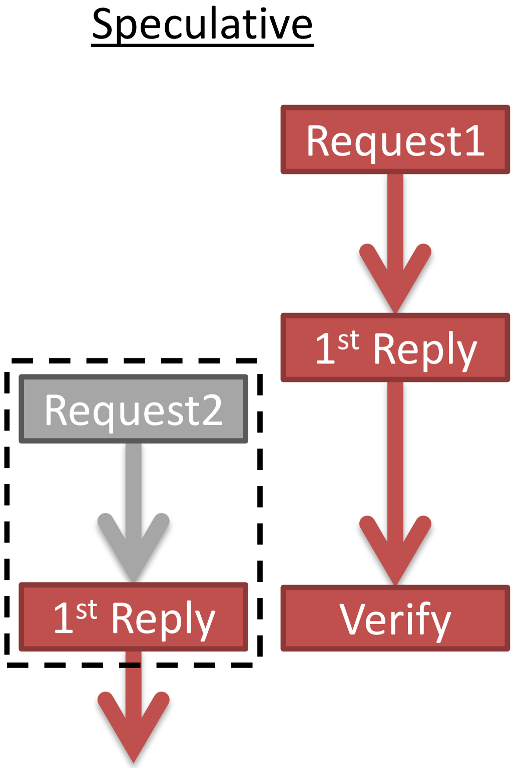
App 3: PBFT-CS Protocol


TIME
Creation Policy: Agree on 1st reply
Send Request
Request
Distributed Replicated Service
(Reply, sig1)
Check reply
Work…
(Reply, sig2)
Check reply
Output Policy: PBFT-CS Messages: Allow

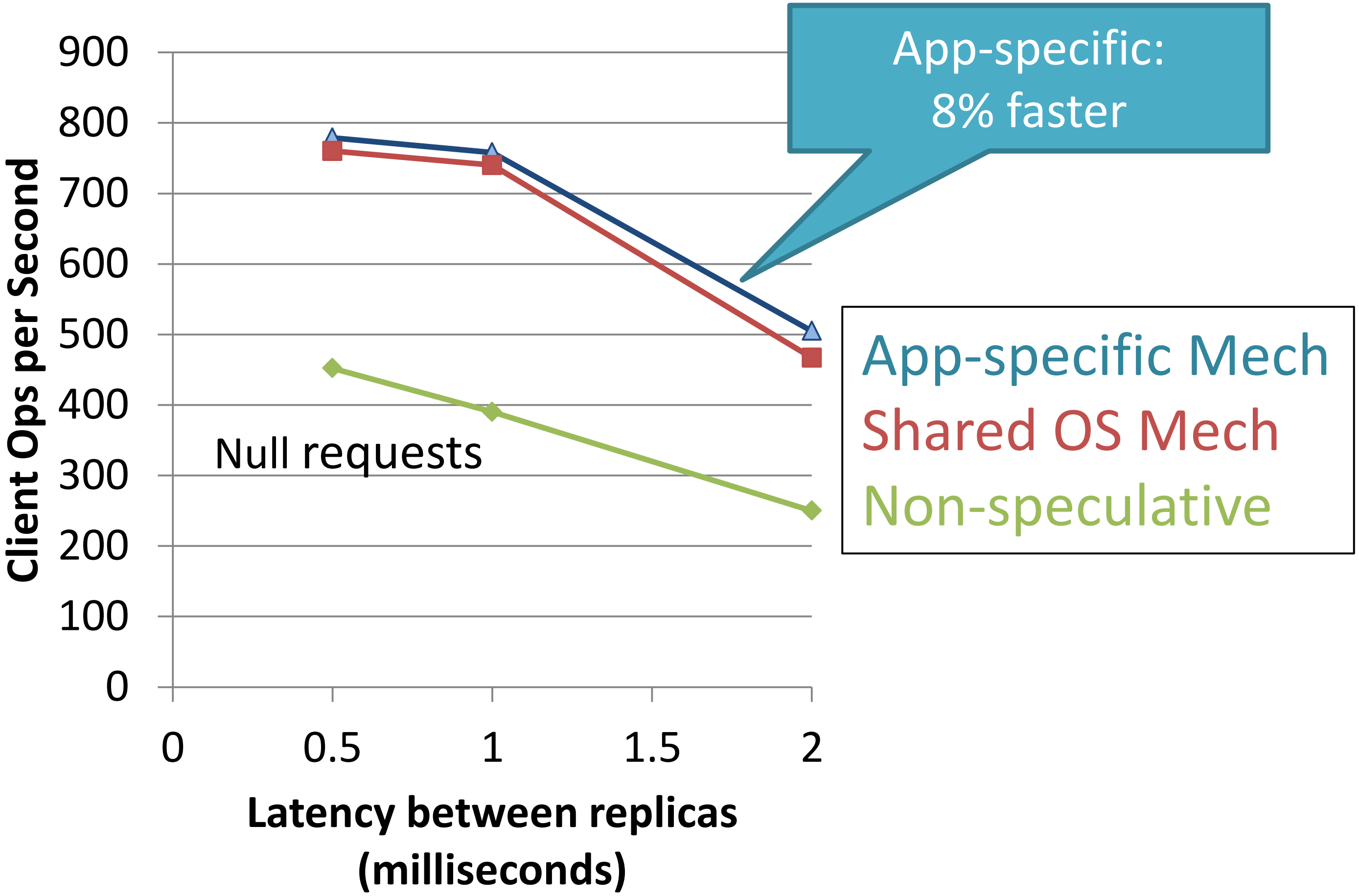
Improved Client Throughput?




1.9x Throughput



Cost of Generic Mechanism



App-specific:
8% faster

Conclusion

- Mechanism
  - Common: checkpoints, output buffering, taint propagation
  - Implemented in OS
- Policy
  - App-specific: Controls creation, output, and commit
  - Implemented in applications
- Demonstrated with 3 case studies
  - Improved parallelism
  - Small overhead relative to app-specific mechanism

Questions?