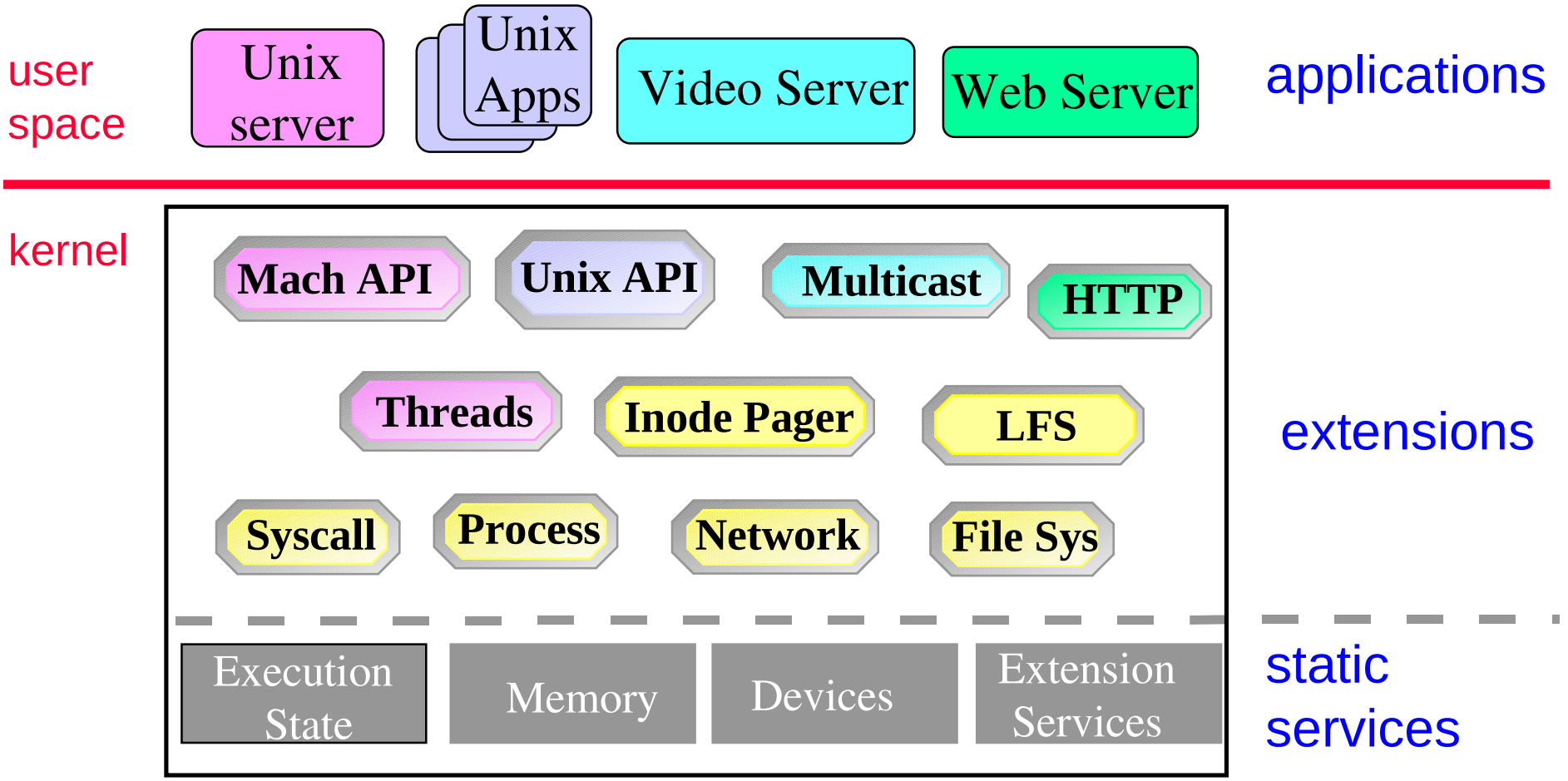
Dynamic Binding for Extensible Systems

Przemysław Pardyak and Brian Bershad

The *SPIN* group
Department of Computer Science and Engineering
University of Washington

The *SPIN* Operating System



Requirements

- Flexible
- Simple
- Fast
- Safe

Existing Approaches

- Dynamic linking
- Service-specific interfaces
- Object-orientation
- Publish-subscribe communications (events)

The *SPIN* Approach

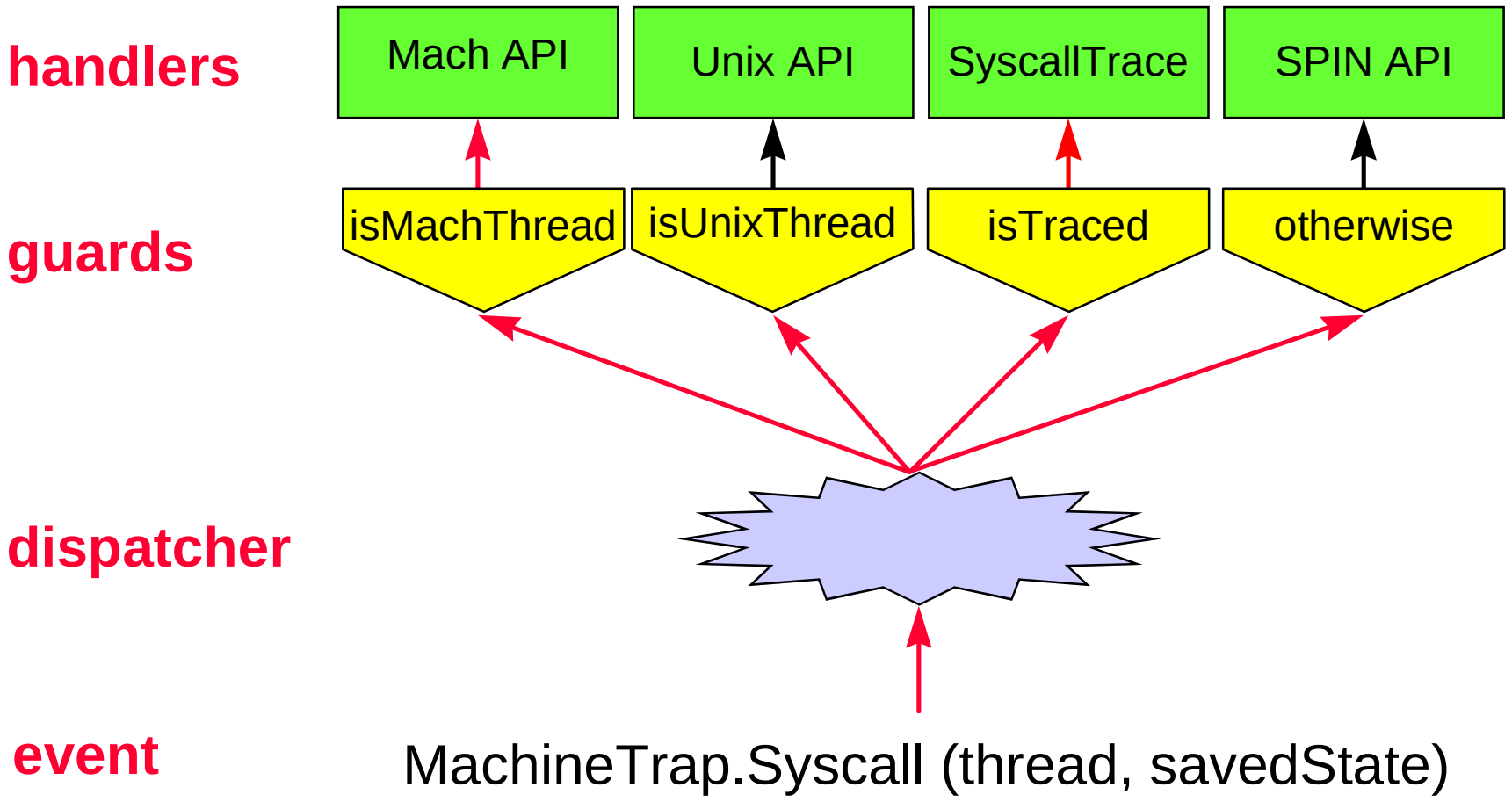
- Combine the best of all worlds
  - Flexibility of publish-subscribe communication
  - Language support for uniformity and simplicity
- Plus
  - Access control for safety
  - Runtime code generation for performance

SPIN Events

EVENTS = PROCEDURES + FLEXIBILITY

- Procedure syntax
- Extended procedure semantics
  - Multicast, conditional execution, asynchrony
- Every procedure is an event

The life of a system call



Programmer's View

Event definition

```
INTERFACE MachineTrap;
PROCEDURE Syscall(
            thread  : Strand.T;
            state   : SavedState);

END MachineTrap.
```

Event invocation

```
MODULE MachineTrapPrivate;

PROCEDURE Syscall(thread, state) =
  BEGIN
    …
    MachineTrap.Syscall(thread,state)
    …
  END Syscall;
END MachineTrapPrivate.
```

Extension

```
MODULE MachEmulation;
PROCEDURE Handler (thread, state) =
  CASE state.v0 OF
      …
    | -65 => VM.vm_allocate();
      …
  END Handler;

PROCEDURE Guard (thread, state)
                    : BOOLEAN =
    RETURN IsMachThread(thread);
  END Guard;

BEGIN
  Dispatcher.InstallHandler(
        MachineTrap.Syscall,
        Guard, Handler);
END MachEmulation.
```

Applicability

- Available in the whole system
- Used to implement
  - System calls
  - Threads
  - Networking
  - VM
- How can an untrusted application use them?

Safety

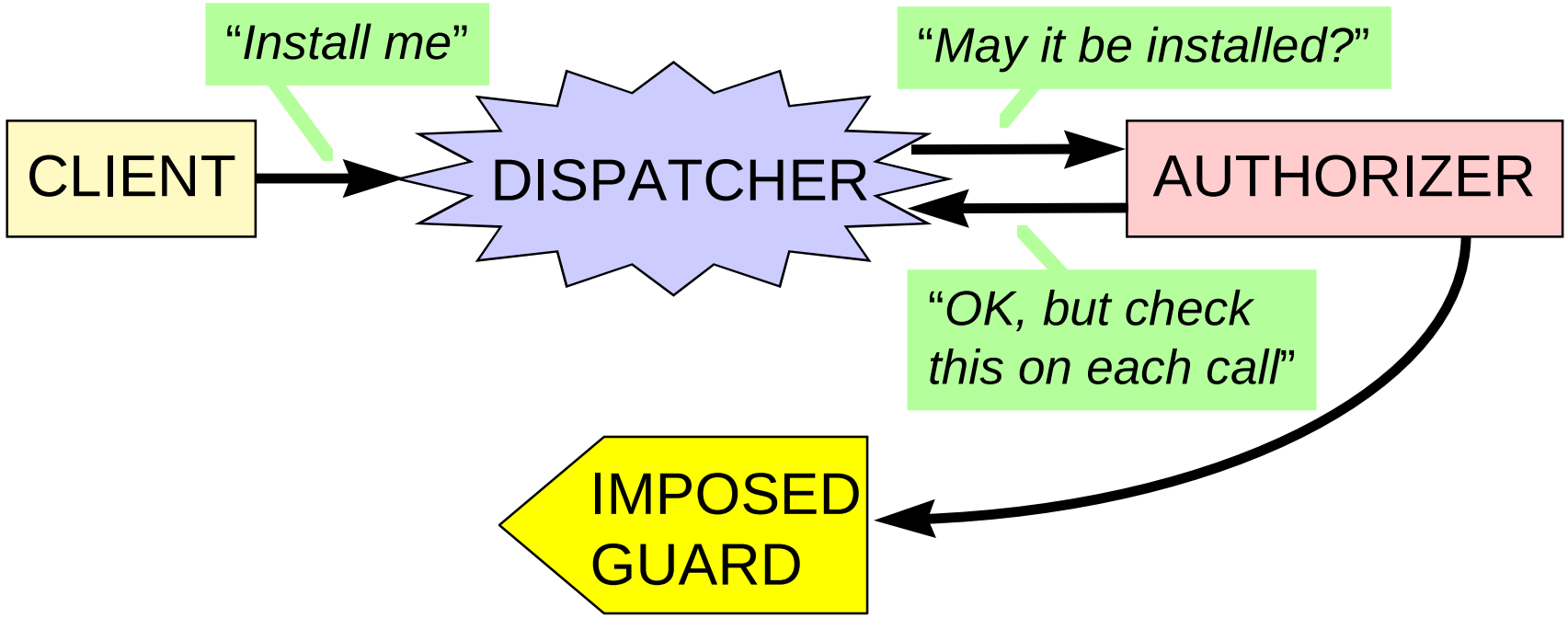
- Events enable communication with untrusted extensions
- Protect
  - Visibility of events
  - Handler installation
  - Event delivery
- Prevent denial of service

Handler installation

- Event's authorizer verifies handler installations

Event delivery

- Imposed guards dynamically check the right to receive the event occurrence



Denial of service

- Problem:
  - A handler may not return in time
- Preventive solution:
  - Call the handler asynchronously
- Remedial solution:
  - Terminate the handler

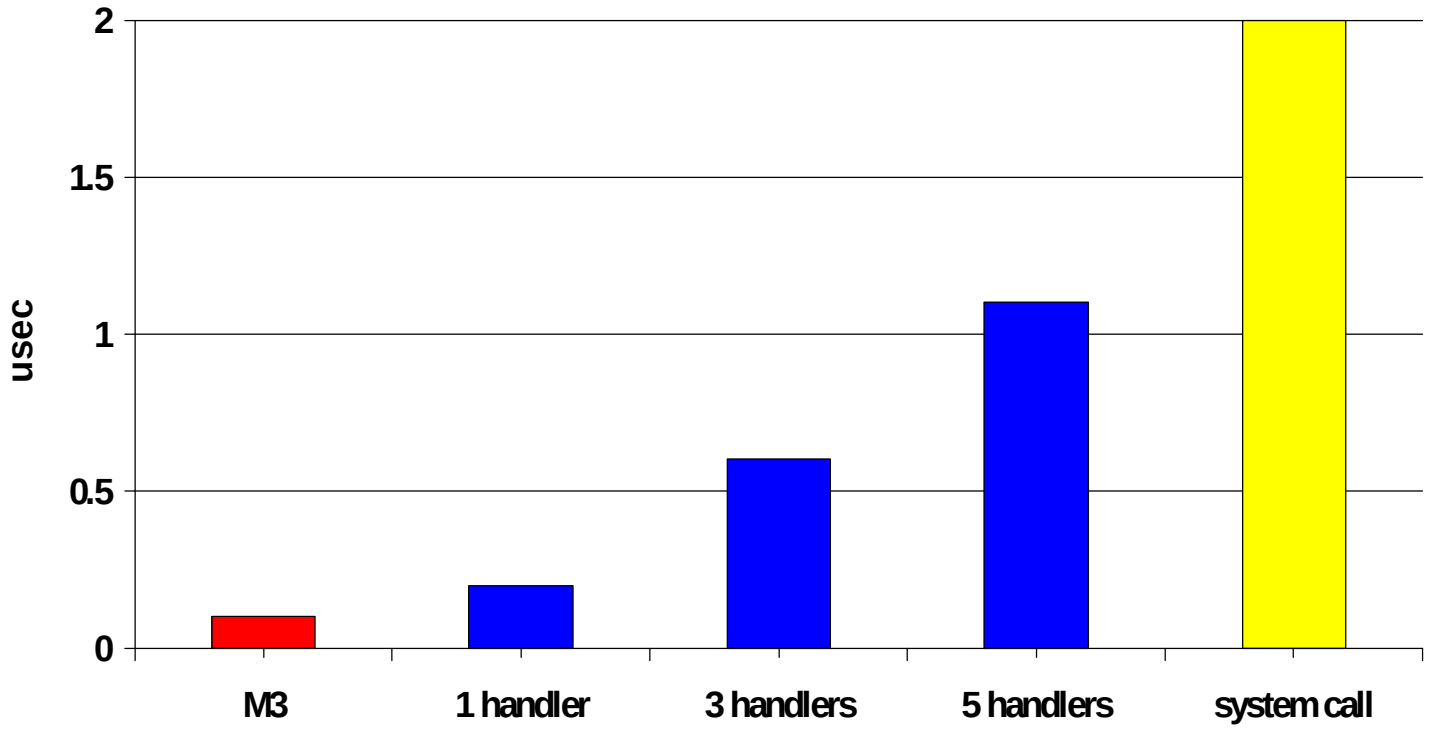
Performance

- The cost of event invocations
- Scalability
- Impact on other parts of the system

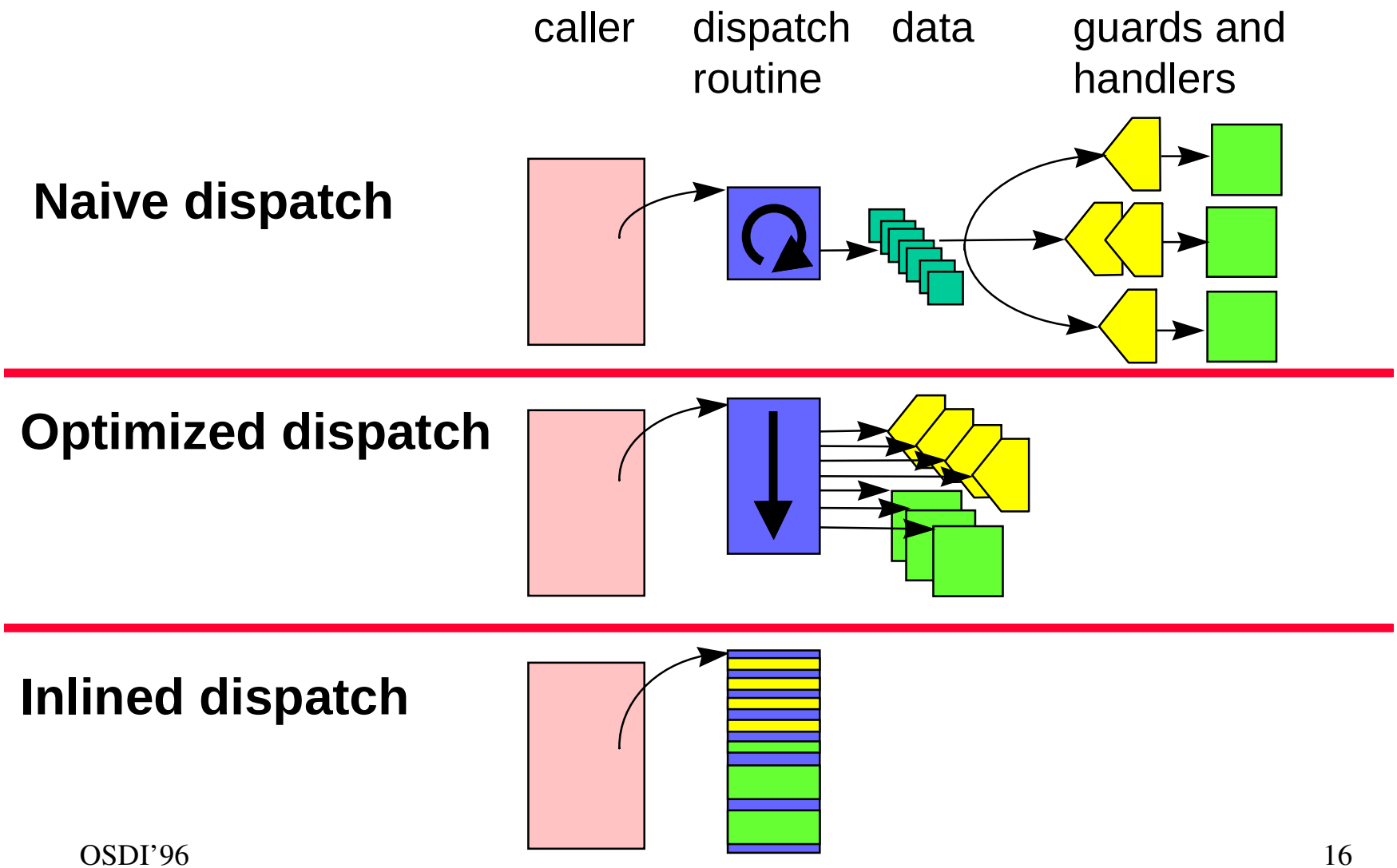
- Measured on DEC Alpha 133MHz AXP 3000/400

The Cost of Event Invocations

- Linear with the number of handlers and guards installed



Implementation



Guard Optimizations

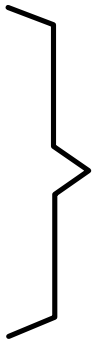
- Opportunity
  - Many guards evaluated, few handlers invoked
- Solution
  - Eliminate redundant tests

Multiple guards:

… pkt.type = Udp ...

… pkt.type = Tcp ...

… pkt.type = Arp ...

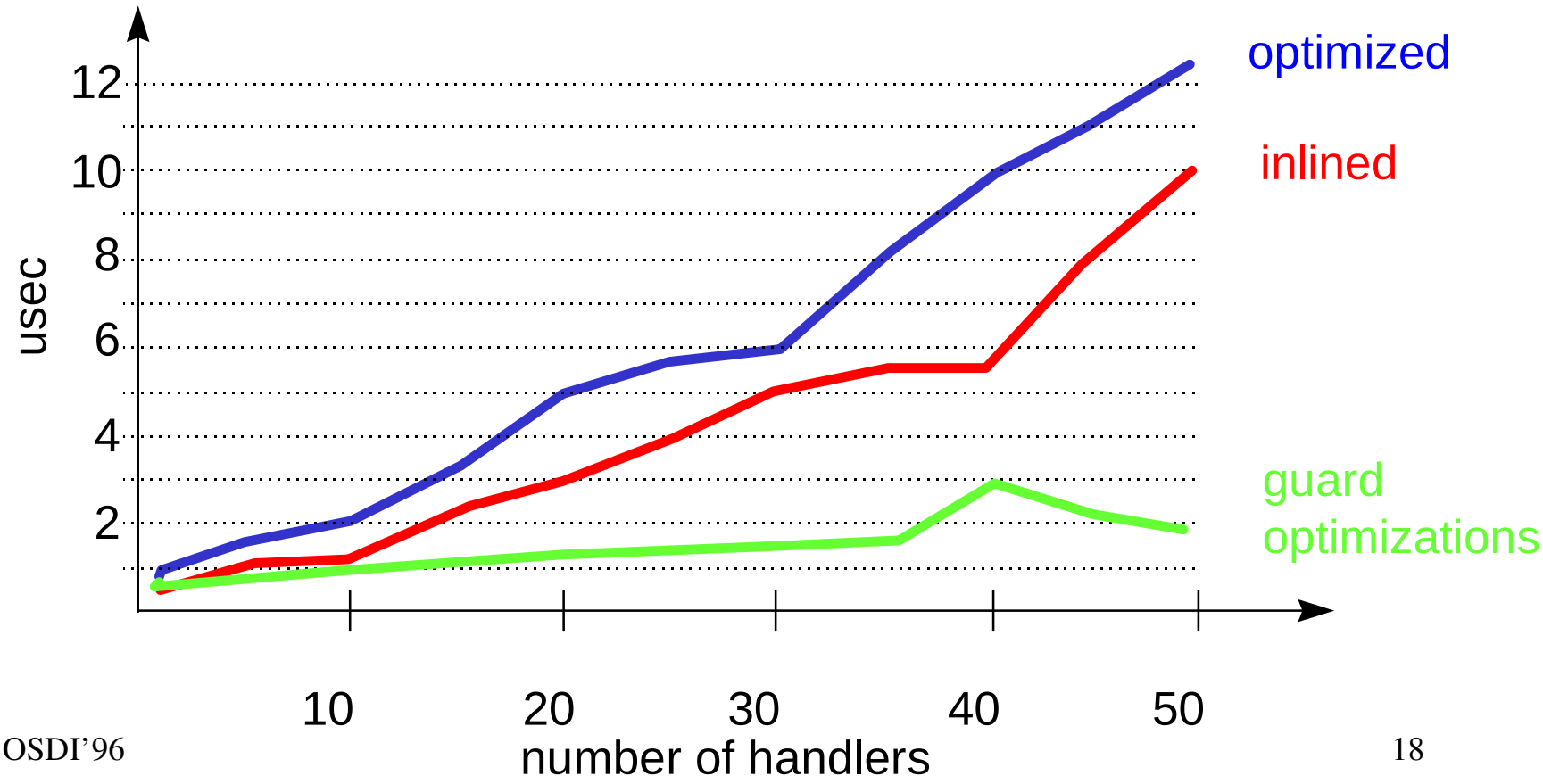


Combined guards:

… Lookup(pkt.type) ...

Scalability

Cost of dispatch as a function of number of handlers when only one handler executes



Conclusions

- Events combine flexible extensibility with the simplicity of a language mechanism
- Extensibility can be cheap and safe