Asynchronous event-driven programming in P

Shaz Qadeer
Microsoft Research

Collaborators:

Ankush Desai	UC Berkeley
Vivek Gupta	Microsoft Windows
Ethan Jackson	Microsoft Research
Sriram Rajamani	Microsoft Research
Damien Zufferey	IST Austria

Event-driven asynchronous programs

- Asynchronous interaction among components
  - at least two components: system and environment
- Events
  - from one component to another

- Asynchrony in a world of devices and services
  - devices connecting to computers
  - clients connecting to services
  - services connecting to services
  - humans connecting to computers

Heisenbugs due to asynchrony and concurrency are difficult to detect, diagnose, and fix!

This project actually started eight years ago…

The modeling conundrum

- “Code ships; models don’t”
- “Why should I model? The code is the model”
- “Models get out-of-sync with the code”
- …

P: modeling or programming?

- Domain-specific language for implementing protocols in event-driven software
  - Blurs the distinction between modeling and programming
  - Systematic exploration of the (nondeterministic) operational semantics

- USB device driver stack in Windows 8 implemented in P

Building Windows 8

An inside look from the Windows engineering team

Building robust USB 3.0 support

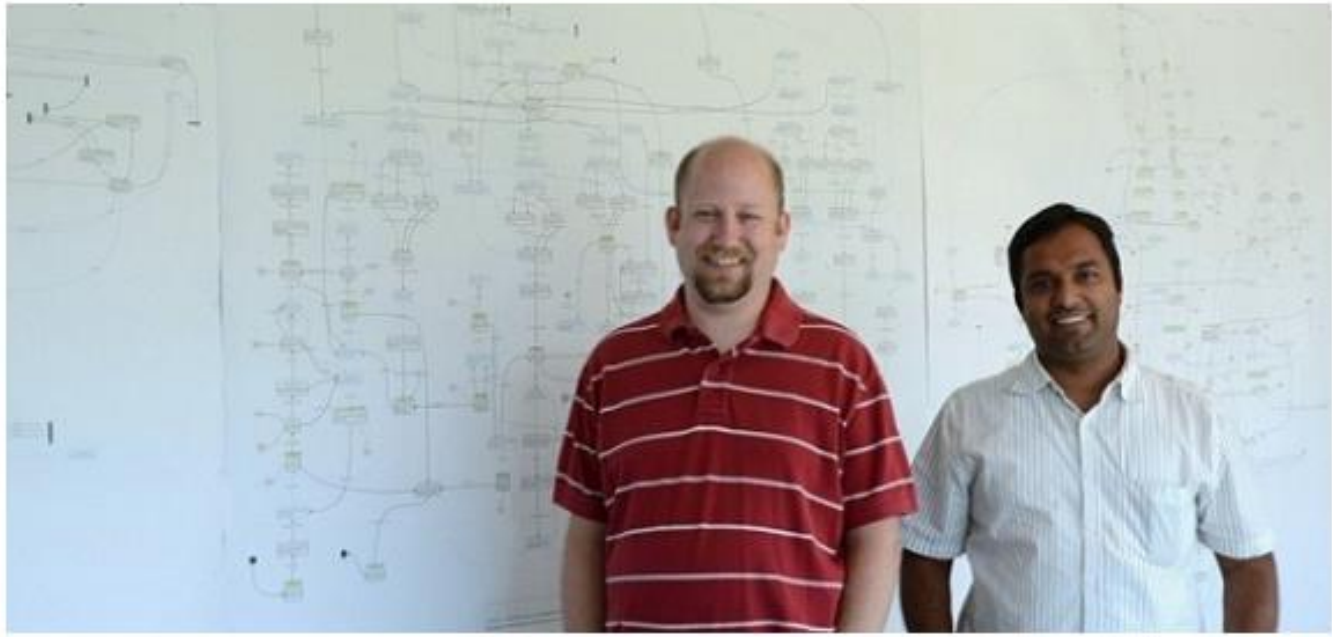
Steven Sinofsky **22 Aug 2011 12:00 PM** 120

RATE THIS

One of the im
for new hardw
at supporting
improvements
not high prod
and available
those file copy
by Dennis Fla
group. –Steve

With throughput up
longer battery life, U
interface. By 2015, al
USB devices will be s

With virtual device development underway, we started designing and prototyping. USB software is complex because it has to manage hubs and devices while still dealing with any errors. To create something with longevity we needed to visualize and document the flow. We designed three massive flow charts and a code generator to automatically convert a Visio diagram into software. Together with Microsoft Research, we refined a tool called Zing, which could validate every aspect of this software model.



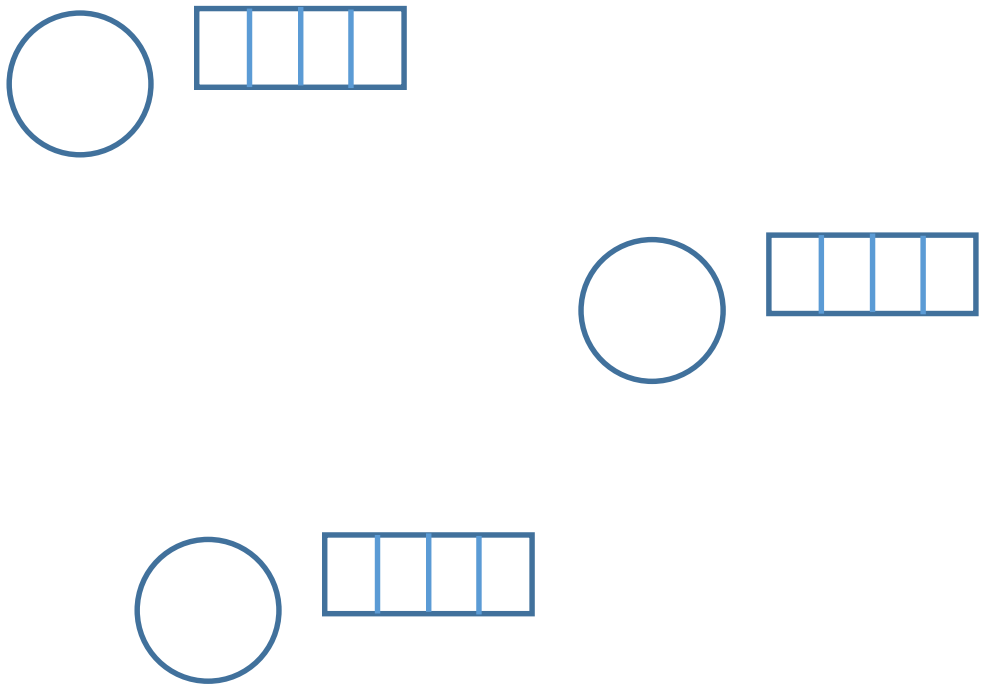
Flow chart with its architects, Randy Aull and Vivek Gupta

Win8 USB development

State Machine	No. of states	No. of transitions	Explored States (millions)	No. of Bugs found
HSM	196	361	5.9	90
PSM 3.0	295	752	1.5	12
PSM 2.0	457	1386	2.2	97
DSM	1919	4238	1.2	120

- Errors discovered and fixed before code starts running
- Precise documentation of interfaces
- Less code but equivalent performance as legacy driver

Communicating state machines



- Remove message from input buffer
- Compute on the local state
- Send messages to other machines

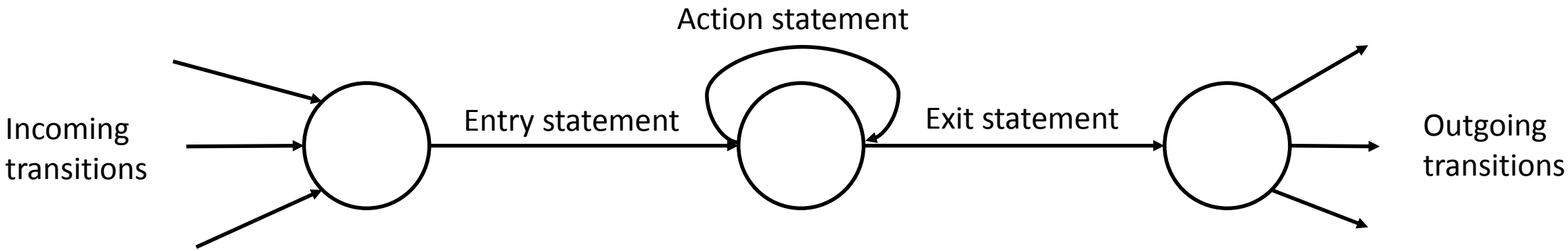
From computational model to programming language

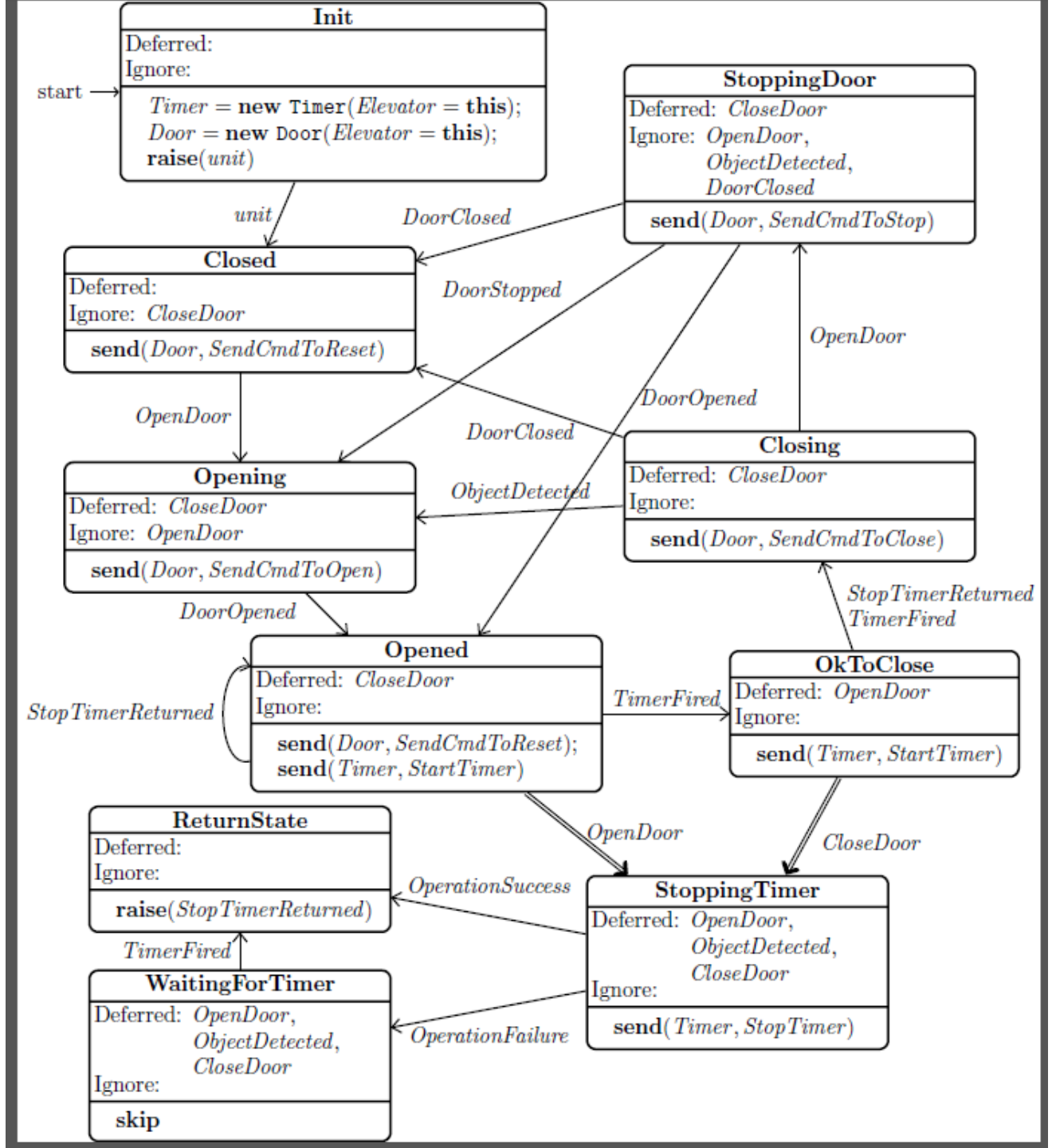
- What are the control and data structures inside a machine?
- How do we write specifications?
- How do we test and verify programs?
- How do we execute programs efficiently?

P language features

- A collection of machine types
  - machine instances are dynamically created
- A collection of events
  - a payload type can be optionally attached to an event
- Each machine contains variables, states, and actions

Control flow inside a state




Init
Deferred:
Ignore:
Timer = new Timer(Elevator = this);
Door = new Door(Elevator = this);
raise(unit)
start
unit
StoppingDoor
Deferred: CloseDoor
Ignore: OpenDoor, ObjectDetected, DoorClosed
send(Door, SendCmdToStop)
DoorClosed
Closed
Deferred:
Ignore: CloseDoor
send(Door, SendCmdToReset)
DoorStopped
OpenDoor
DoorOpened
DoorClosed
Closing
Deferred: CloseDoor
Ignore:
send(Door, SendCmdToClose)
OpenDoor
Opening
Deferred: CloseDoor
Ignore: OpenDoor
send(Door, SendCmdToOpen)
ObjectDetected
DoorOpened
StopTimerReturned
TimerFired
Opened
Deferred: CloseDoor
Ignore:
send(Door, SendCmdToReset);
send(Timer, StartTimer)
StopTimerReturned
TimerFired
OkToClose
Deferred: OpenDoor
Ignore:
send(Timer, StartTimer)
OpenDoor
CloseDoor
ReturnState
Deferred:
Ignore:
raise(StopTimerReturned)
OperationSuccess
StoppingTimer
Deferred: OpenDoor, ObjectDetected, CloseDoor
Ignore:
send(Timer, StopTimer)
TimerFired
WaitingForTimer
Deferred: OpenDoor, ObjectDetected, CloseDoor
Ignore:
skip
OperationFailure

Hierarchy within a machine

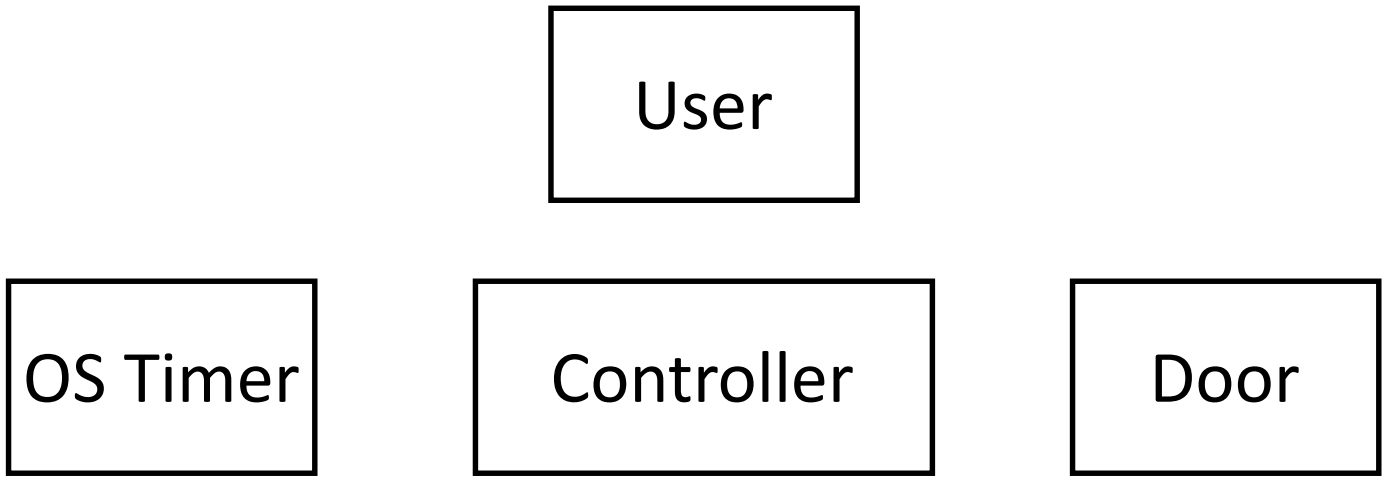
- Reuse functionality implemented by a collection of states
- Call/Return semantics
- Deferred events and installed actions are passed from caller to callee

Specification and Validation

Correctness of a P program

- Default specifications
  - Safety: every dequeued event is handled
  - Liveness: every enqueued event is eventually dequeued under fair scheduling
- Bounds
  - maximum queue size for machine types
  - maximum number of instances for event types
- Assertions

The elevator and its environment

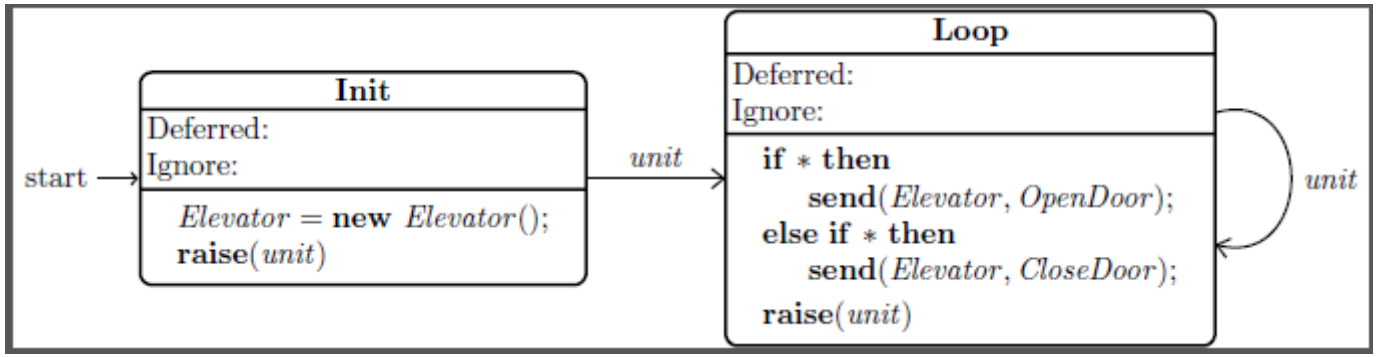


Ghosts

- A machine type or a variable can be declared ghost

- Ghost machines and variables are used for modeling and specification
  - computational model no different from real machines

User ghost machine



Validation

- Verification of P programs is undecidable
  - dynamic machine creation, unbounded buffers, etc.

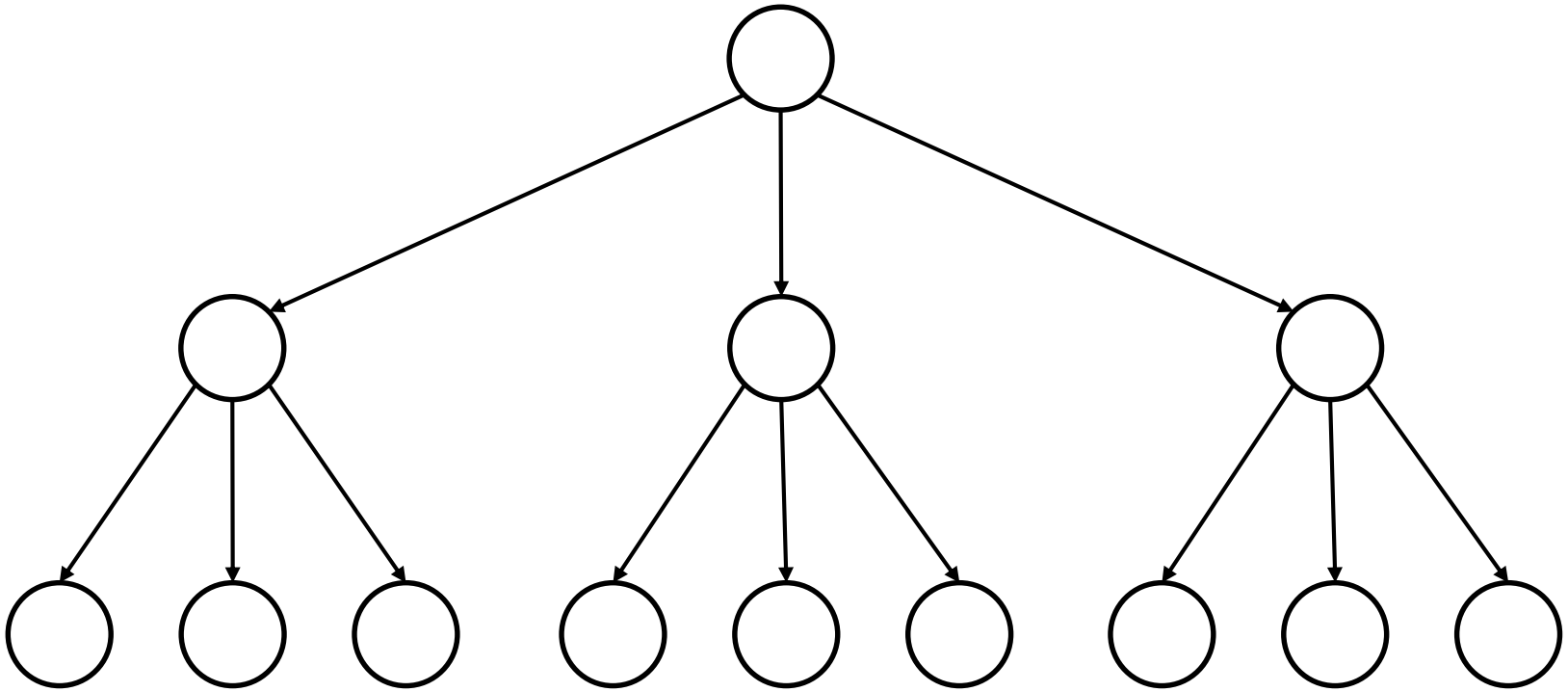
- Exploration of the (nondeterministic) operational semantics
  - Explicit choice
  - Implicit scheduling choice

- Implemented via translation to Zing
  - depth-first search with state deltas
  - efficient state fingerprinting
  - parallel search

Searching efficiently

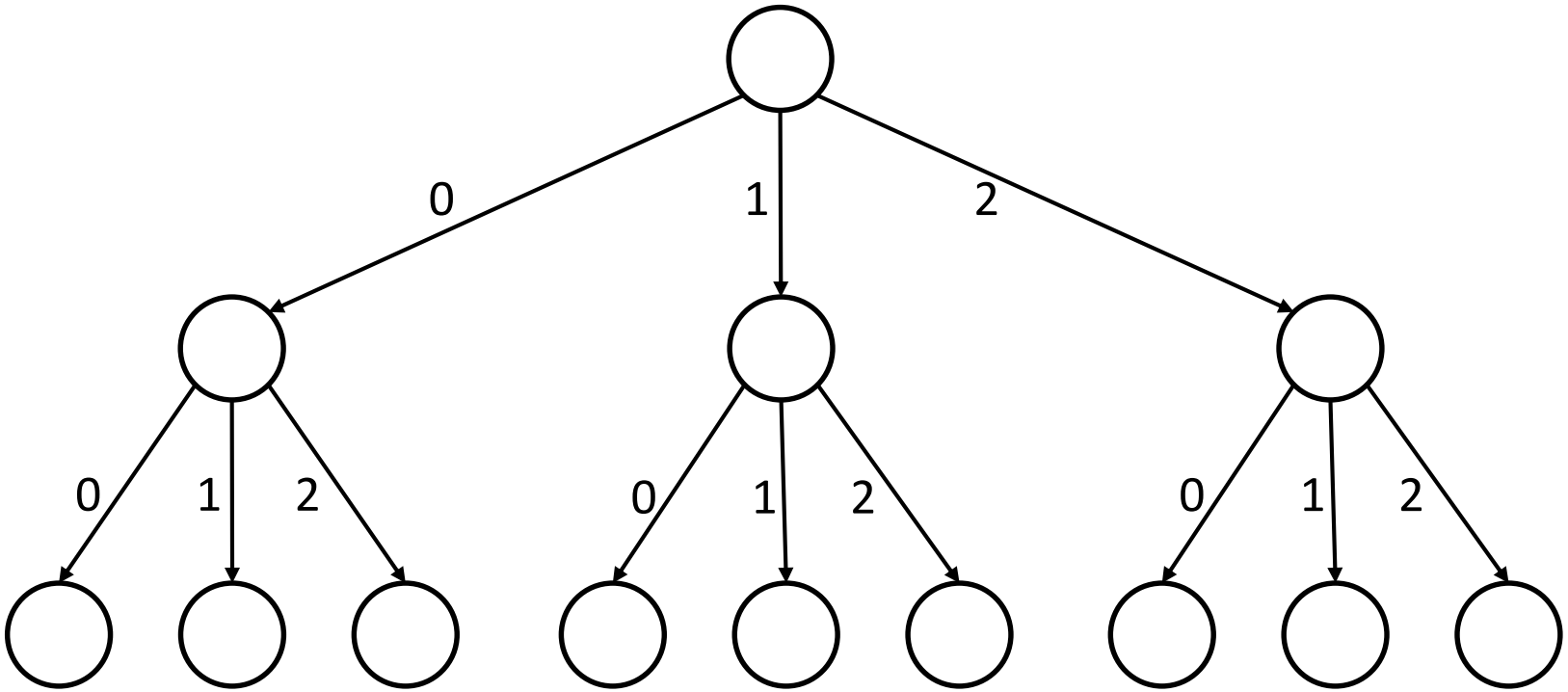
- Exploit commutativity
  - local operations commute in either direction
  - dequeue commutes forward in time
- Iterative deepening with execution length
- Iterative deepening with delaying schedulers

Standard view of state-transition graph



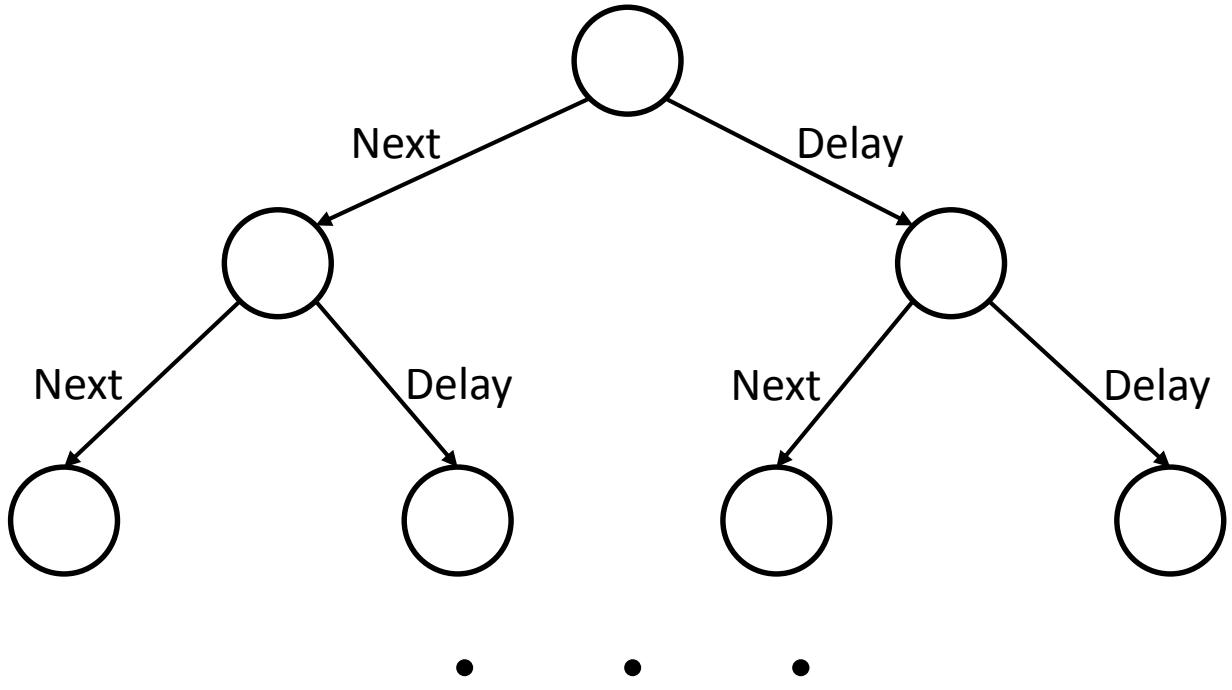
. . .

Alternate view of state-transition graph



. . .

Searching with a delaying scheduler



I scheduler invocations
d delays
executions = I^d

Stateless schedulers
- Process order
- Random order

Stateful schedulers
- Round robin
- Run to completion

Iterative deepening with delaying schedulers

- Explore sub-graphs with increasing cost iteratively using frontiers
  - avoid caching scheduler state
- Both reachability and cycle detection
- General framework allowing any delaying scheduler to be plugged in

Compiler and Runtime

Erasure by typing ghosts

- Compiler erases ghost elements when generating code for execution

- Information flow is only from real to ghost world
  - real machine cannot use nondeterministic choice
  - cannot assign ghost value to a real variable
  - cannot send ghost value to a real machine
  - cannot assign real machine id to a ghost variable

Compiler

- Converts the program into a collection of C data structures
  - machine table
  - state table
  - action table
  - transition table

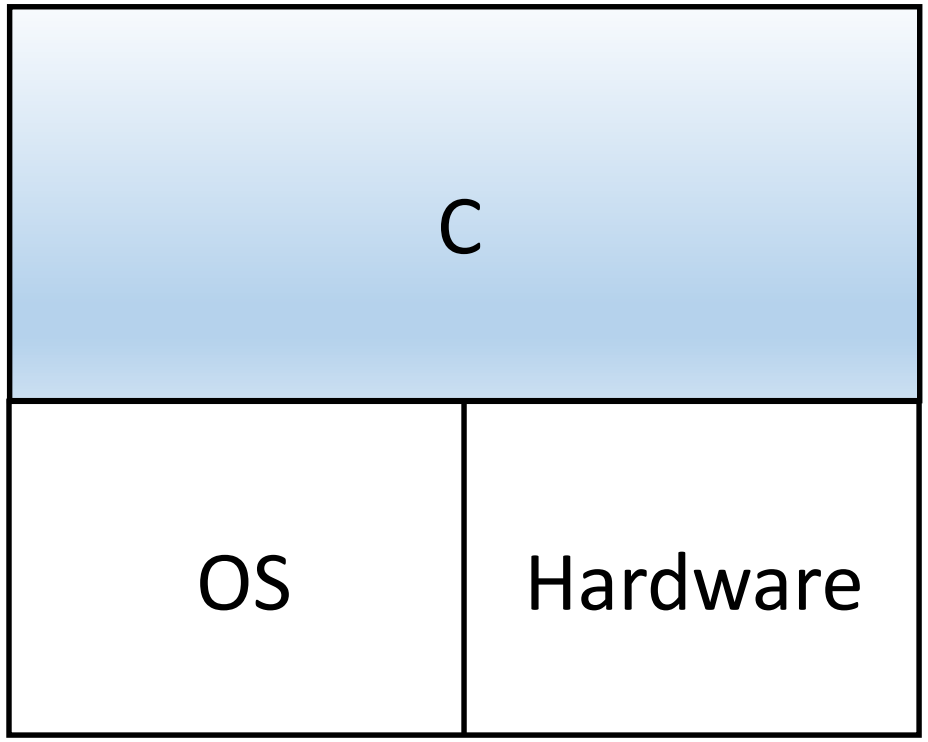
- Runtime accesses these data structures to execute state machines

Runtime

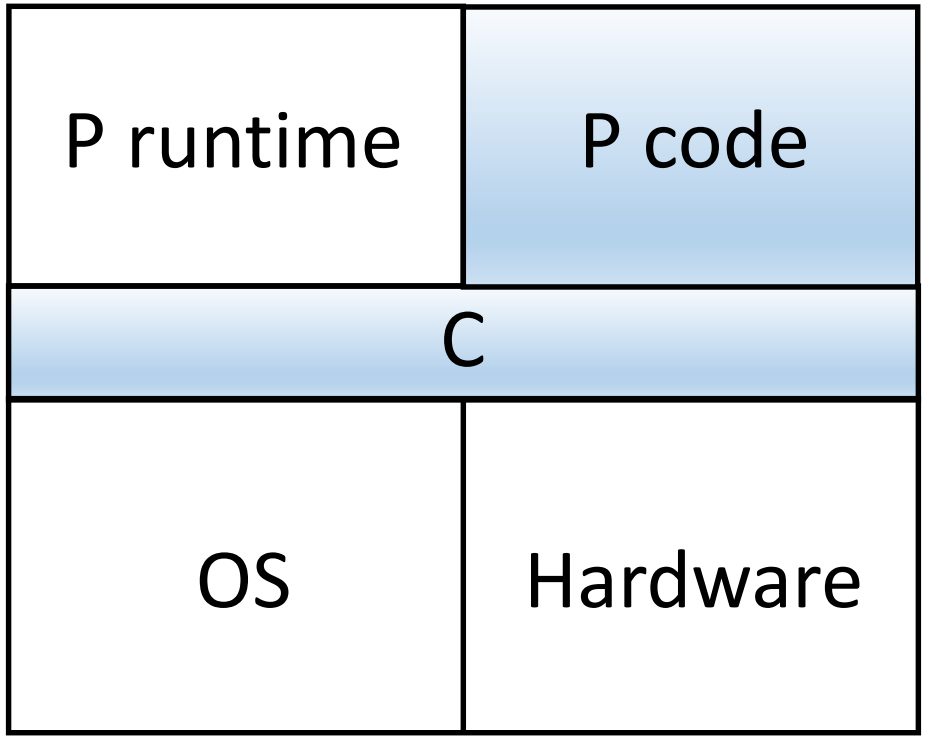
- Provides
  - machine creation and deletion
  - input buffer management
  - execution of machines
  - foreign function interface

- Reactive event-driven computation piggybacked on external threads
  - external thread enqueuing a message quiesces all state machines
  - locking for coordination among multiple external threads

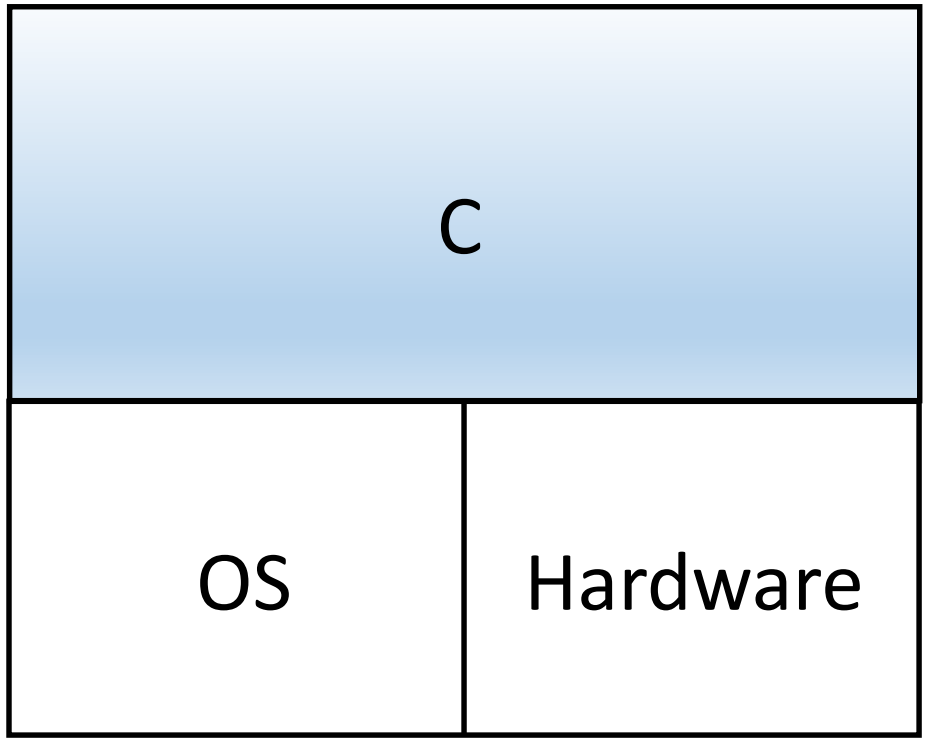
Without P



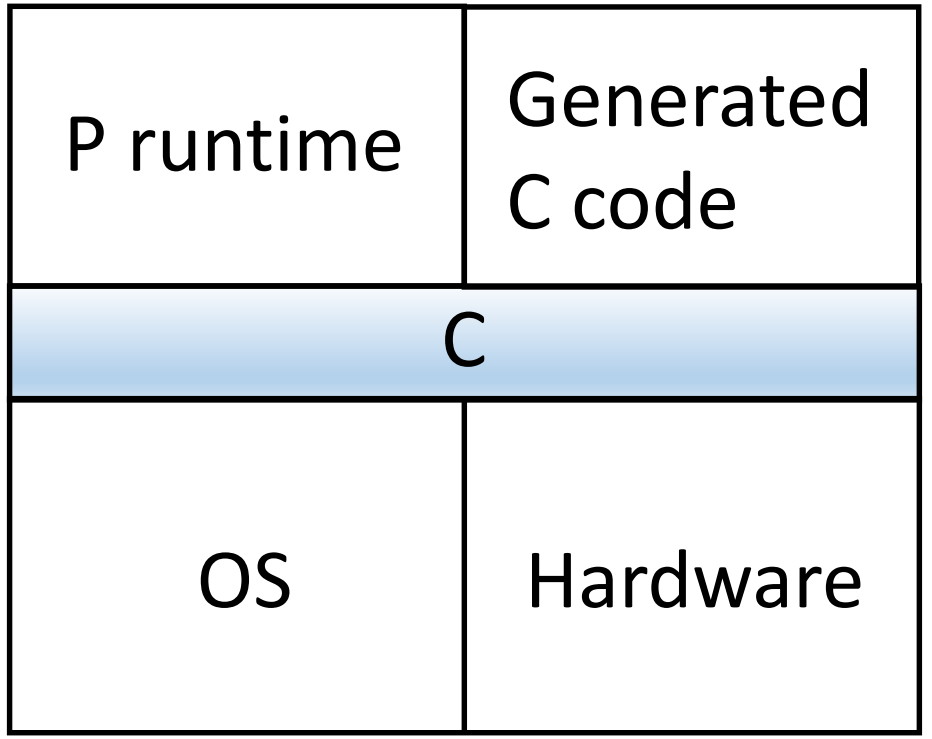
With P



Without P



With P



Status of P

- Full system in place
  - Compiler to C and Zing
  - Runtime for drivers
  - Visual editor for P programs
- In use for driver development in Windows and Windows Phone
- Beyond drivers
  - Asynchronous services
  - Implementations of distributed protocols