# 98-023A : Concurrent and Distributed Programming w/ Inferno and Limbo

Phillip Stanley-Marbell
pstanley@ece.cmu.edu

# Lecture Outline

- Brief Review of CSP
- Introduction to SPIN and Promela

# Syllabus

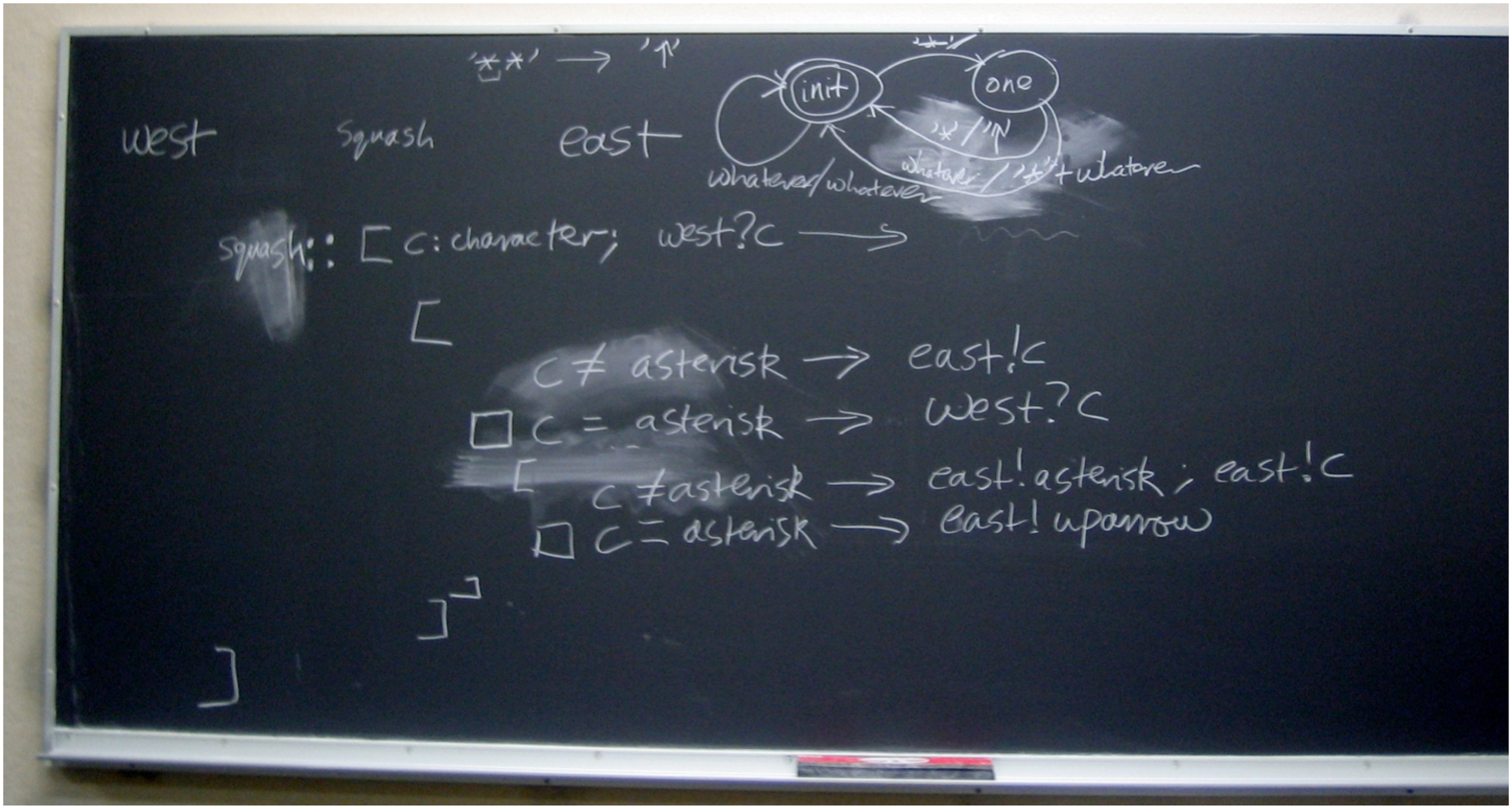
- Week 1: Introduction to Inferno
- Week 2: Overview of the Limbo programming language
- Week 3: Types in Limbo
- Week 4: Inferno Kernel Overview
- Week 5: Inferno Kernel Device Drivers
- Week 6: NO CLASS
- Week 7: C applications as resource servers: Built-in modules and device drivers
- Week 8: Case study I — building a distributed multi-processor simulator
- Week 9: Platform independent Interfaces: Limbo GUIs; Project Update
- Week 10: Programing with threads, CSP
- Week 11: Debugging concurrent programs; Promela and SPIN
- Week 12: Factotum, Secstore and Inferno's security architecture
- Week 13: Case study II — Edisong, a distributed audio synthesis and sequencing engine

# CSP Review: Coroutines

- Squash in CSP

```
X :: *[c: character; west?c →
  [c ≠ asterisk → east!c □ c = asterisk → west?c;
    [c ≠ asterisk → east!asterisk; east!c
    □ c = asterisk → east!uparrow]
    ]
]
```

# CSP Review: Coroutines



# Example: Coroutines

- Squash in Limbo

```
for (;;)
{
  c :=<- west;
  case c
  {
    asterisk =>
      c =<- west;
      case c
      {
        asterisk =>
          east <-= uparrow;
        * =>
          east <-= asterisk;
          east <-= c;
      }
    * =>
      east <-= c;
  }
}
```

Squash in CSP (again)

```
X ::*[

c: character; west?c →


  [c = asterisk →
    west?c;
    [
      c = asterisk →
      east!uparrow

    □ c ≠ asterisk →
      east!asterisk;
      east!c
    ]

□ c ≠ asterisk → east!c
  ]
]
```

# Designing Concurrent Applications

- Something to think about : Why is it often said that "the FreeBSD networking stack is good" ?
  - Designing concurrent applications is hard
    - Many different things happening at the same time (obviously)
    - Difficult even with a firm specification of desired behavior
    - Even worse, what is a "firm specification" ??
- In many cases, people "tune" the implementation of complex concurrent applications until they "get it right"

# Designing Concurrent Applications : *Specification*

- How do you specify behaviors ?
  - You'll need to be able to specify the primitive operations
  - Primitive operations in real programs (recall CSP lectures): *Computation*, *Sequencing*, *Choice*, *Repetition* and *I/O*
- How do you specify the concurrent composition of behaviors
  - You'll need to be able to specify how to create instances of different behaviors
  - Specify how instances are interleaved or that you don't care, and let system choose random interleavings

[From "Tutorial: Design and Validation of Protocols" G. J. Holzmann, 1991]

The protocol defines a simplex data-transfer channel, i.e., it will pass data in only one direction, from sender to receiver. Control information, however, flows in both directions. It is assumed that the system has perfect error detection.

To each message sent from A to B we attach an extra bit called the alternation bit. After B receives the message it decides if the message is error-free. It then sends back to A a verification message, consisting of a single verify bit, indicating whether or not the immediately preceding A to B message was error-free. After A receives this verification, one of three possibilities hold:

1. The A to B message was good
2. The A to B message was bad
3. A cannot tell if the A to B message was good or bad because the verification message (sent from B to A) was in error

In cases 2 and 3 A resends the same A to B message as before. In case 1 A fetches the next message to be sent, and sends it, inverting the setting of the alternation bit with respect to the previous A to B message.

Whenever B receives a message that is not in error it compares the alternation bit of this new message to the alternation bit of the most recent error-free reception. If the alternation bits are equal the new message is not accepted. The new message is accepted only if the two alternation bits differ. The verification messages from B to A indicate error-free reception independently of the acceptance of the messages.

Initialization of this scheme depends upon A and B agreeing on an initial setting of the alternation bit. This is accomplished by an A to B message whose error-free reception (but not necessarily acceptance) forces B's setting of the alternation bit. Multiple receptions of such a message cannot do harm.

This protocol has the property that every message fetched by A is received error-free at least once and accepted at most once by B.

# Promela

- Promela is a language for modeling systems to be verified by the SPIN model checker
- Promela is a *specification language*
  - You use it to describe a behavior (e.g., a protocol)
  - Lets you specify computation (very basic), sequencing, choice and I/O
  - Language primitives are very similar to primitives provided by CSP

# SPIN

- SPIN is a tool for automatically validating Promela specifications

- SPIN : Simple Promela INterpreter

- Has a large amount of theory behind it that we won't go into

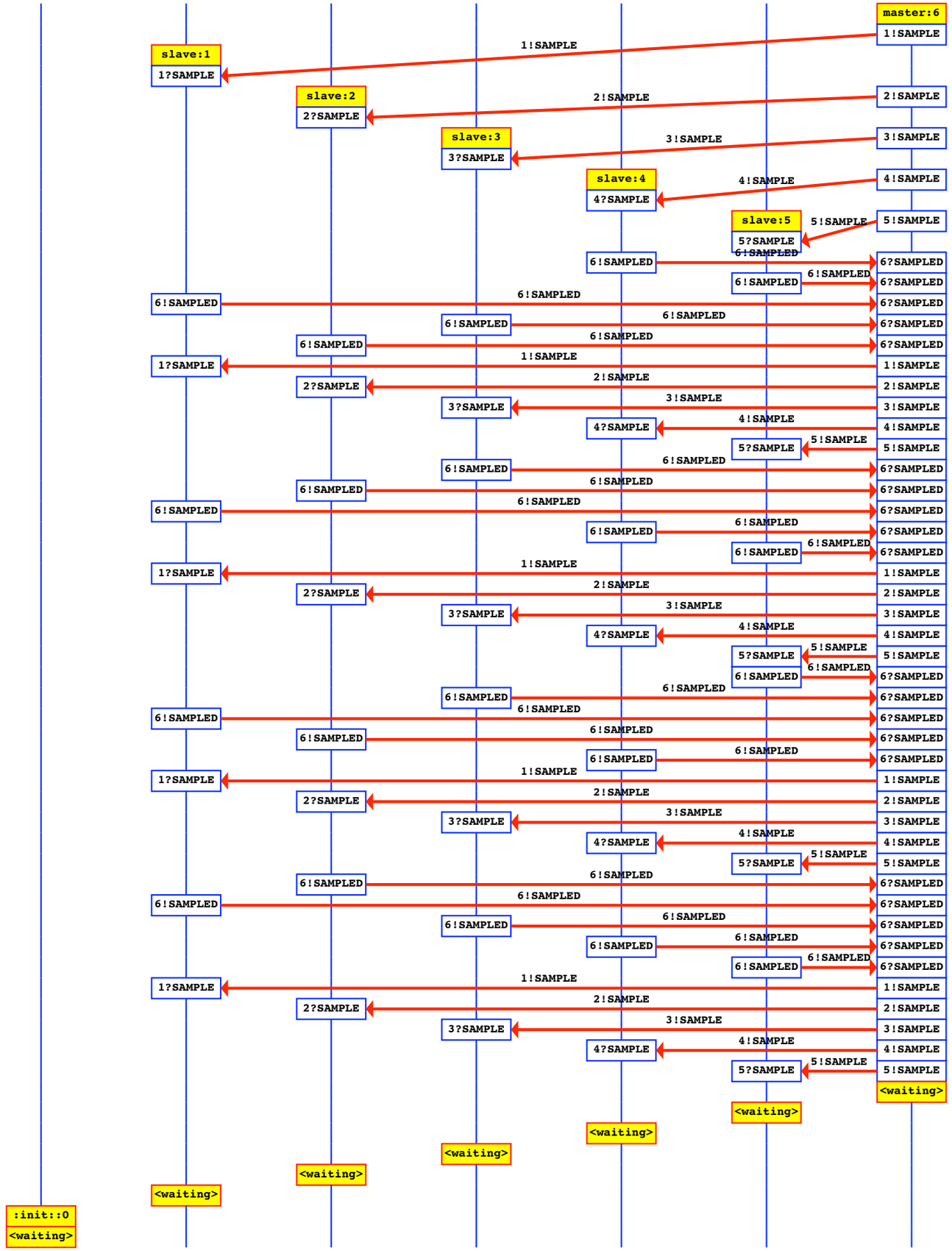
# Example

```
#define NUMCHANS        6
#define NUMSLAVES       5
#define SAMPLEDCHAN     5

mtype = {SAMPLE, SAMPLED};
chan netseg[NUMCHANS] = [0] of {mtype};

proctype slave(byte my_id)
{
        byte   got_sample;

        got_sample = 0;

        do
        :: netseg[my_id]?SAMPLE -> got_sample = 1;

        :: got_sample -> netseg[SAMPLEDCHAN]!SAMPLED;
                                got_sample = 0;
        od
}

proctype master()
{
...
}

init
{
        run slave(0); run slave(1);
        run slave(2); run slave(3);
        run slave(4); run master();
}
```

# Using SPIN

- Perform random simulation of behavioral model
  - SPIN can perform random or guided simulation of the behavioral model, with random interleavings of the different concurrent behaviors in model
- Verify properties specified in Linear Temporal Logic (LTL)
  - Properties and invariants can be specified as LTL formulae
  - Scenarios where these properties can be violated will be caught and example activations shown

# Message Sequence Chart



# SPIN Demo

- You can do everything from the command line
- **Xspin** is a graphical front end to spin

# Handouts

- "Communicating Sequential Processes"; C.A.R. Hoare, *Communications of the ACM*, Volume 21, Number 8, 1978
- "Using SPIN"; Gerard J. Holzmann, *Plan 9 Documents, Volume 2*, available on the web at `http://plan9.bell-labs.com/sys/doc/index.html`

# Next

- Next lecture: More depth on Promela and SPIN ( ABP protocol example)
- Next week: Inferno's security architecture

*Fin.*