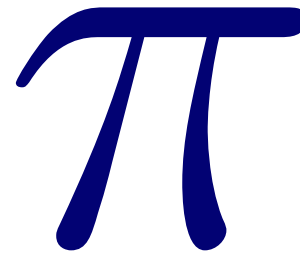


# $\pi$ -Calculus for Fun & Profit

Morgan Deters  
mdeters@cse.wustl.edu

---



[www.cs.wustl.edu/~doc/](http://www.cs.wustl.edu/~doc/)

Spring 2004 Seminar on  
Software Systems

5 April 2004

# Outline

Not a theory talk

# Outline

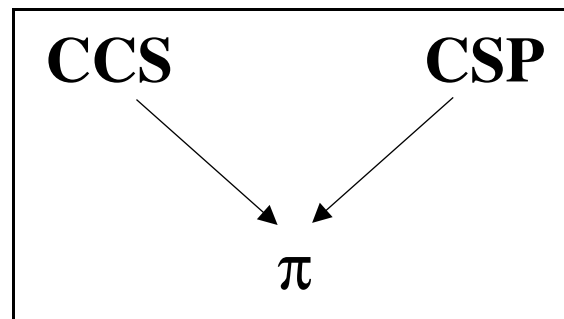
$\pi$  *tutorial*

$\pi$  *design*

$\pi$  *analysis*

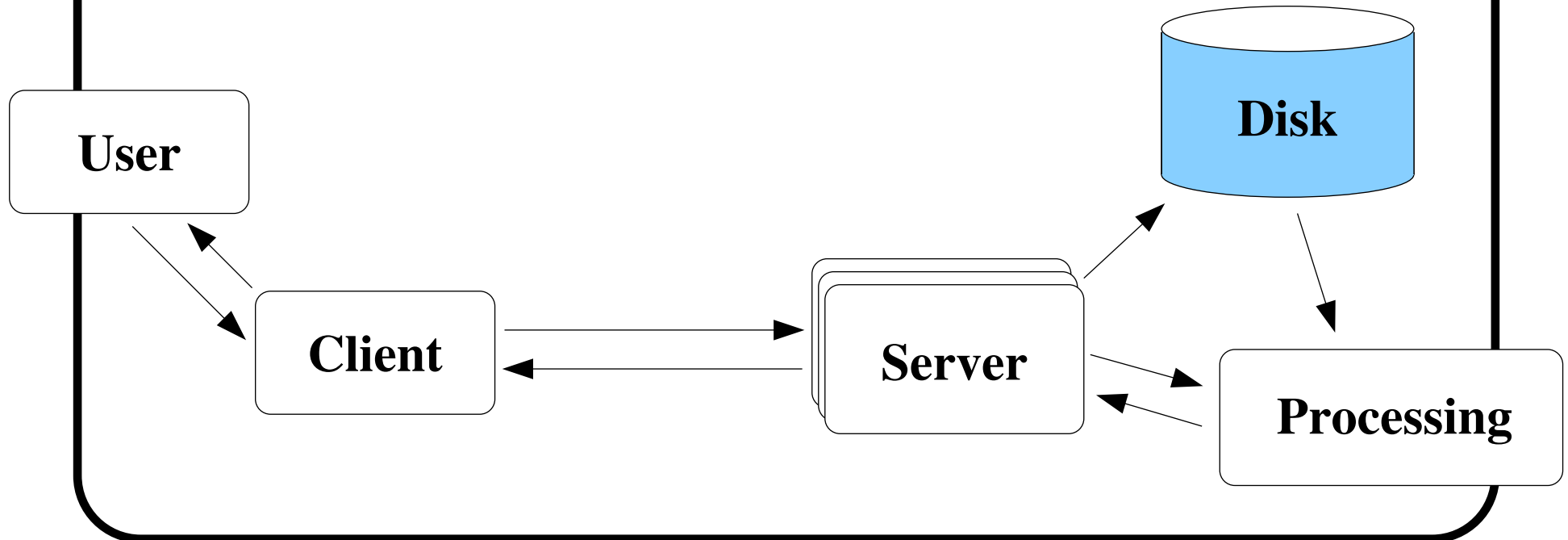
# Introduction

- Intended to model concurrent systems
  - $\lambda$  not good (natural) enough
  - K.I.S.S.
- *Processes* communicate via *channels*
- *Channels* are named, *processes* are anonymous
- Computation asynchronous, communication synchronized
- Computationally complete



# $\pi$

- Names ( $a$ ) and co-names ( $\bar{a}$ ) react
- Names can be passed through reaction
- Processes evolve through reaction
  - can spin off others conditionally or unconditionally
- Basis for mobile design as well



# Syntax

$$P ::= 0 \mid P + P \mid \tau.P \mid a.A \mid \bar{a}.C \mid P \mid P \mid$$

$$\text{if } a = b \text{ then } P \text{ else } P \mid \underbrace{\nu a.P}_{\text{(binds } a \text{ in } P)} \mid D(\vec{a})$$

(binds  $a$  in  $A$ )

$$A ::= \underbrace{(a)A}_{\text{(binds } a \text{ in } A)} \mid P$$

$$C ::= [a]C \mid \underbrace{\nu a.C}_{\text{(binds } a \text{ in } C)} \mid P$$

$a, b, c$  are names;  $\vec{a}$  is a vector  $a_1, \dots, a_n$ ;  $\tau$  is internal

$$a(b_1, \dots, b_n).P \stackrel{\text{def}}{=} a.(b_1) \cdots (b_n)P$$

$$\nu b_1, \dots, b_n. \bar{a}[c_1, \dots, c_m].P \stackrel{\text{def}}{=} \nu b_1 \cdots \nu b_n. \bar{a}.[c_1] \cdots [c_m]P$$

\* Syntax from [Dam]

# Informally Speaking...

$a.P$	transition $a$ leaves you at $P$
$P + Q$	choice
$a.P \mid \bar{a}.Q$	reaction
$P \mid Q$	concurrency
if $\phi$ then $P$ else $Q$	conditional
$\nu a.P$	$a$ is bound as a private name
$a(b)$	reception
$\bar{a}[b]$	transmission

# Transition Rules

$$\begin{array}{l}
 \text{SUM} \quad \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'} \qquad \text{PRE} \quad \frac{\cdot}{\alpha.P \xrightarrow{\alpha} P} \\
 \text{COM} \quad \frac{P \xrightarrow{\nu \vec{b}. \vec{a}[\vec{b}']} P' \quad Q \xrightarrow{a(\vec{c})} Q'}{P | Q \xrightarrow{\tau} \nu b.(P' | (Q' \{\vec{b}'/\vec{c}\}))} \qquad \text{PAR} \quad \frac{P \xrightarrow{\alpha} P'}{P | Q \xrightarrow{\alpha} P' | Q} \\
 \text{IF}_1 \quad \frac{P \xrightarrow{\alpha} P'}{\text{if } a = a \text{ then } P \text{ else } Q \xrightarrow{\alpha} P'} \\
 \text{IF}_2 \quad \frac{Q \xrightarrow{\alpha} Q'}{\text{if } a = b \text{ then } P \text{ else } Q \xrightarrow{\alpha} Q'} \quad (a \neq b) \\
 \text{RES} \quad \frac{P \xrightarrow{\alpha} P'}{\nu a.P \xrightarrow{\alpha} \nu a.P'} \quad (a \notin \text{names}(\alpha)) \\
 \text{OPEN} \quad \frac{P \xrightarrow{\nu \vec{b}. \vec{a}[\vec{b}']} P'}{\nu c.P \xrightarrow{\nu \vec{b}. c. \vec{a}[\vec{b}']} P'} \quad (c \neq a, c \in \vec{b}' \setminus \vec{b}) \\
 \text{ID} \quad \frac{P\{\vec{b}'/\vec{a}\} \xrightarrow{\alpha} P'}{D(\vec{b}) \xrightarrow{\alpha} P'} \quad (D(\vec{a}) \stackrel{\text{def}}{=} P)
 \end{array}$$

\* Transition rules from [Dam]

# Semantics... by example!

$$\mathit{Coroutine}(a) = \nu b.\bar{a}[b].b(c).\mathit{Coroutine}(c)$$

$$\mathit{Buf}_1(i, o) = i(a).\bar{o}[a].\mathit{Buf}_1(i, o)$$

$$\mathit{Buf}_{n_1+n_2}(i, o) = \nu m.(\mathit{Buf}_{n_1}(i, m) \mid \mathit{Buf}_{n_2}(m, o))$$

$$\mathit{Buf}(i, o) = i(a).\nu m.(\mathit{Buf}(i, m) \mid \bar{o}[a].\mathit{Buf}(m, o))$$

$$\mathit{BufCell}(o, d, n_l, n_r) = \bar{o}[d].\bar{n}_l[o, n_r].0 + n_r(o', n).\mathit{BufCell}(o', d, n_l, n)$$

$$\begin{aligned} \mathit{StartCell}(i, o, n) = & i(d).\nu o'.\nu n'.(\mathit{StartCell}(i, o', n') \mid \\ & \mathit{BufCell}(o, d, n', n)) \\ & + n(o', n').\mathit{StartCell}(i, o', n') \end{aligned}$$

$$\mathit{GCBuf}(i, o) = \nu n.\mathit{StartCell}(i, o, n)$$

\* Examples from [Dam]

# Mobile Telephones Example

[Milner 1991]

# Representing Definitions

[Milner 1991]

- Definitions aren't "special"
  - can be done within  $\pi$

- Given a definition:

$$D(\vec{x}) \stackrel{\text{def}}{=} P \quad (fn(P) \subseteq \{\vec{x}\})$$

- Let  $\hat{P}$  be  $P$  with all recursive invocations  $D(\vec{x})$  replaced by  $!d(\vec{x}).\hat{P}$ .
- Replace outer invocations of  $D(\vec{z})$  with  $\nu d.(\bar{d}[\vec{z}] \mid !d(\vec{z}).\hat{P})$
- Translation is weakly congruent to original form

# Boolean-Valued Names

(from [Milner 1999])

$$\begin{aligned} \text{True}(l) &\stackrel{\text{def}}{=} l(t, f).\bar{t} \\ \text{False}(l) &\stackrel{\text{def}}{=} l(t, f).\bar{f} \end{aligned}$$

- So, if you have a “boolean location”  $b$ , you can test it:
  - $\nu t.\nu f.\bar{b}[t, f].(t.doSomething + f.doSomethingElse)$
  - $\nu t.\nu f.\bar{b}[t, f].\nu resume.($   
 $t.doSomething.\overline{resume} +$   
 $f.doSomethingElse.\overline{resume}) \mid$   
 $resume.resumedComputation$
- These are *ephemeral*
- To make *persistent*, use replication

# Church Encoding

(from [Dam])

- In  $\lambda$ , represent  $n \in \mathbb{N}$  as function application
  - **zero**:  $\lambda f.\lambda x.x$
  - **succ**( $n$ ):  $\lambda f.\lambda x.f(nfx)$
  - **7** is  $\lambda f.\lambda x.f(f(f(f(f(fx))))))$
- In  $\pi$ , represent  $n \in \mathbb{N}$  as reaction

$$\begin{aligned} \text{Zero}(n) &\stackrel{\text{def}}{=} n(s, z).\bar{z} \\ \text{Succ}(n, m) &\stackrel{\text{def}}{=} m(s, z).\bar{s}[n] \end{aligned}$$

- **one = 1** defined by  $\nu \text{zero}.\text{Zero}(\text{zero}) \mid \text{Succ}(\text{zero}, \text{one})$
- **read  $n$  with**  $\nu s.\nu z.\bar{n}[s, z].(s(m).\text{recurse} + z.\text{base})$

# $\pi$ Lists

(from [Milner 1999])

$$\begin{aligned} Nil &\stackrel{\text{def}}{=} (k).k(n, c).\bar{n} \\ Cons(CAR, CDR) &\stackrel{\text{def}}{=} (k).\nu a.\nu d.(k(n, c).\bar{c}[a, d] \mid \\ &\quad CAR[a] \mid CDR[d]) \end{aligned}$$

- To represent a list  $[x_1, \dots, x_n]$ , build one as usual:  
 $Cons(x_1, Cons(\dots Cons(x_n, Nil) \dots))$
- These are ephemeral
  - to make persistent, replicate  $k$  abstractions

# $\pi$ Update

(from [Milner 1999])

$$\begin{aligned}
 \text{Nullref}(r) &\stackrel{\text{def}}{=} r(n, c).(\bar{n}.\text{Nullref}[r] + \\
 &\quad c(v').\text{Ref}[r, v'] + n.\text{Nullref}[r]) \\
 \text{Ref}(r, v) &\stackrel{\text{def}}{=} r(n, c).(\bar{c}[v].\text{Ref}[r, v] + \\
 &\quad c(v').\text{Ref}[r, v'] + n.\text{Nullref}[r])
 \end{aligned}$$

- The value held in *Ref* is mobile
- To instantiate:  $\text{Store}[Ctor]$

$$\text{Store}(V) \stackrel{\text{def}}{=} (r).\nu v.(\text{Ref}[r, v] \mid V[v])$$

- To read  $r$ :  $\nu n.\nu c.\bar{r}[n, c].(c(v).P + n.Q)$
- To write  $r \leftarrow v$ :  $\nu n.\nu c.\bar{r}[n, c].\bar{c}[v]$
- To nullify  $r$ :  $\nu n.\nu c.\bar{r}[n, c].\bar{n}$

# $\pi$ Objects

(David Walker, as recounted in [Milner 1999])

- Translate class declaration for class  $C$ :

$$!\nu c.\overline{klass}_C[c].Object_C[c]$$

where

$$\begin{aligned} Object_C(a) &\stackrel{\text{def}}{=} \\ &\nu v_1 \cdots \nu v_n.(Nullref[v_1] \mid \cdots \mid Nullref[v_n] \mid !Methods_C[a]) \\ Methods_C(a) &\stackrel{\text{def}}{=} \nu m_1 \cdots \nu m_n.\bar{a}[m_1 \cdots m_n]. \\ &\quad (m_1(\llbracket args \rrbracket).\llbracket code \rrbracket + \cdots + m_n(\llbracket args \rrbracket).\llbracket code \rrbracket)) \end{aligned}$$

# $\pi$ Typing

- $\pi$  process errors

- communication errors:

$$a(m, n) \mid \bar{a}[x] \not\rightarrow_{\text{(error)}}$$

- detection undecidable

- ◊ see *e.g.* [Vasconcelos & Ravara 1998]

- So admit a sound but incomplete type system

- based on number/type of arguments

*e.g.*

$$\left\{ \begin{array}{l} \text{TALK} \mapsto \epsilon \\ \text{SWITCH} \mapsto \text{TALK}, \text{SWITCH} \\ \text{GAIN} \mapsto \text{TALK}, \text{SWITCH} \\ \text{LOSE} \mapsto \text{TALK}, \text{SWITCH} \end{array} \right.$$

# $\pi$ for Design

- Asynchronous, communicating processes
- Natural for distributed/concurrent computation
- Macromodules symposium... *except*
  - level of abstraction
  - “grand-unified” process hierarchy
    - ◊ hardware
    - ◊ kernel
    - ◊ middleware
    - ◊ applications
- Nowick’s talk

# $\pi$ for Analysis

- Correctness
- Independence/Containment
- Prove deadlock isn't possible

# References

- [**Dam**] Dam, Mads. Proof systems for pi-calculus logics. To appear in de Queiroz (ed.), *Logic for Concurrency and Synchronisation*, Studies in Logic and Computation, Oxford Univ Press, 2003(?).
- [**Lin 1994**] Lin, H. Symbolic bisimulations and proof systems for the pi-calculus. Technical Report 1994:07, COGS, University of Sussex, 1994.
- [**Milner 1991**] Milner, Robin. *The Polyadic  $\pi$ -Calculus: a Tutorial*. Technical Report ECSLFCS-91-180, Computer Science Department, University of Edinburgh, UK, October 1991.  
<http://citeseer.ist.psu.edu/milner91polyadic.html>
- [**Milner 1999**] Milner, Robin. *Communicating and mobile systems: the  $\pi$ -calculus*. Cambridge University Press, 1999.
- [**Vasconcelos & Ravara 1998**] Vasconcelos, Vasco T., and Ravara, António. *Communication Errors in the  $\pi$ -Calculus are Undecidable*.  
<http://www.di.fc.ul.pt/~vv/papers/commerrors.pdf>, 1998.

# Discussion

`www.cs.wustl.edu/~doc/`



Washington  
University in St. Louis

Spring 2004 Seminar on  
Software Systems

5 April 2004