Reasoning about concurrency:
interference requires permission

Aaron Turon, Mitchell Wand

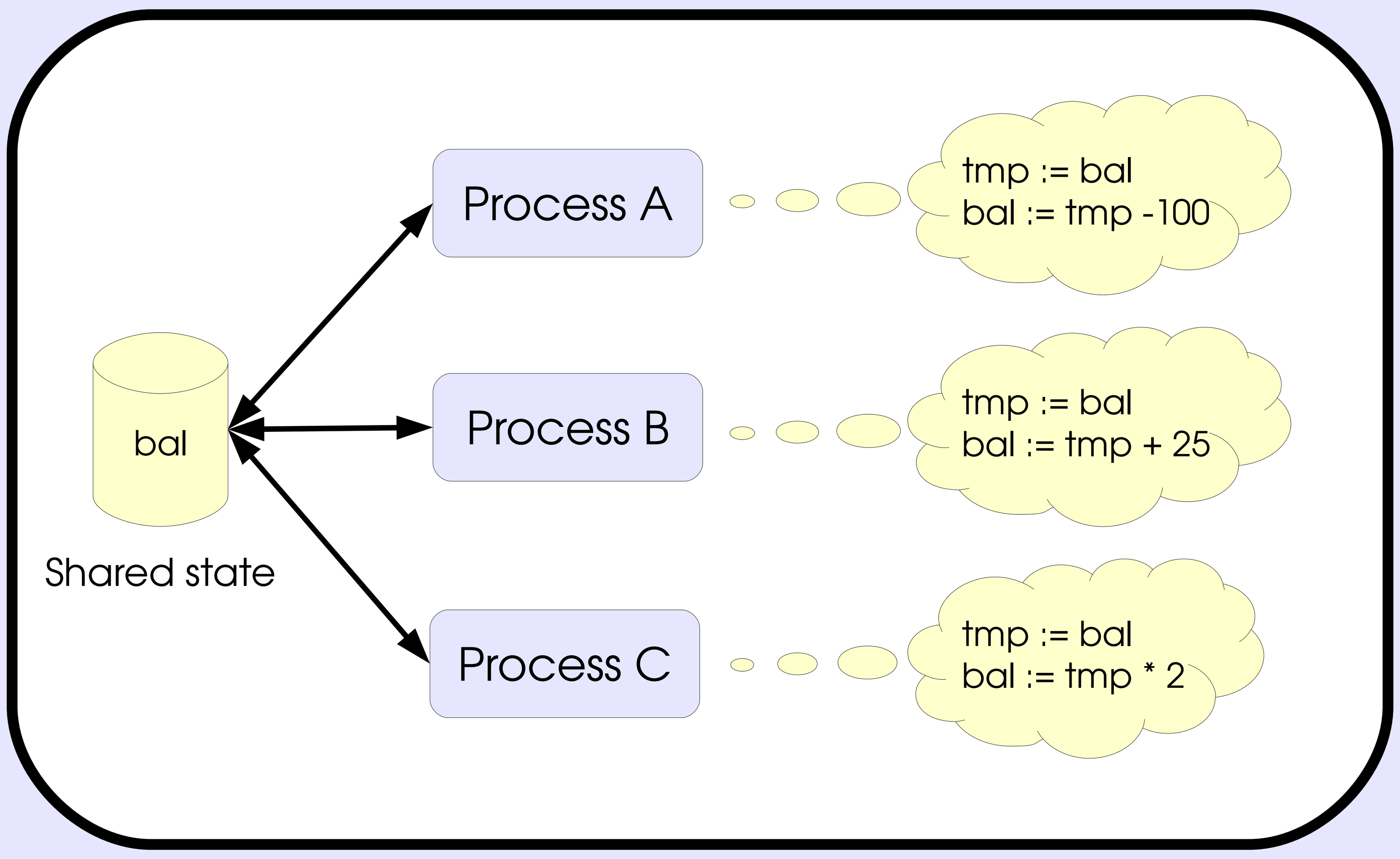

Figure 1: Shared-state concurrency

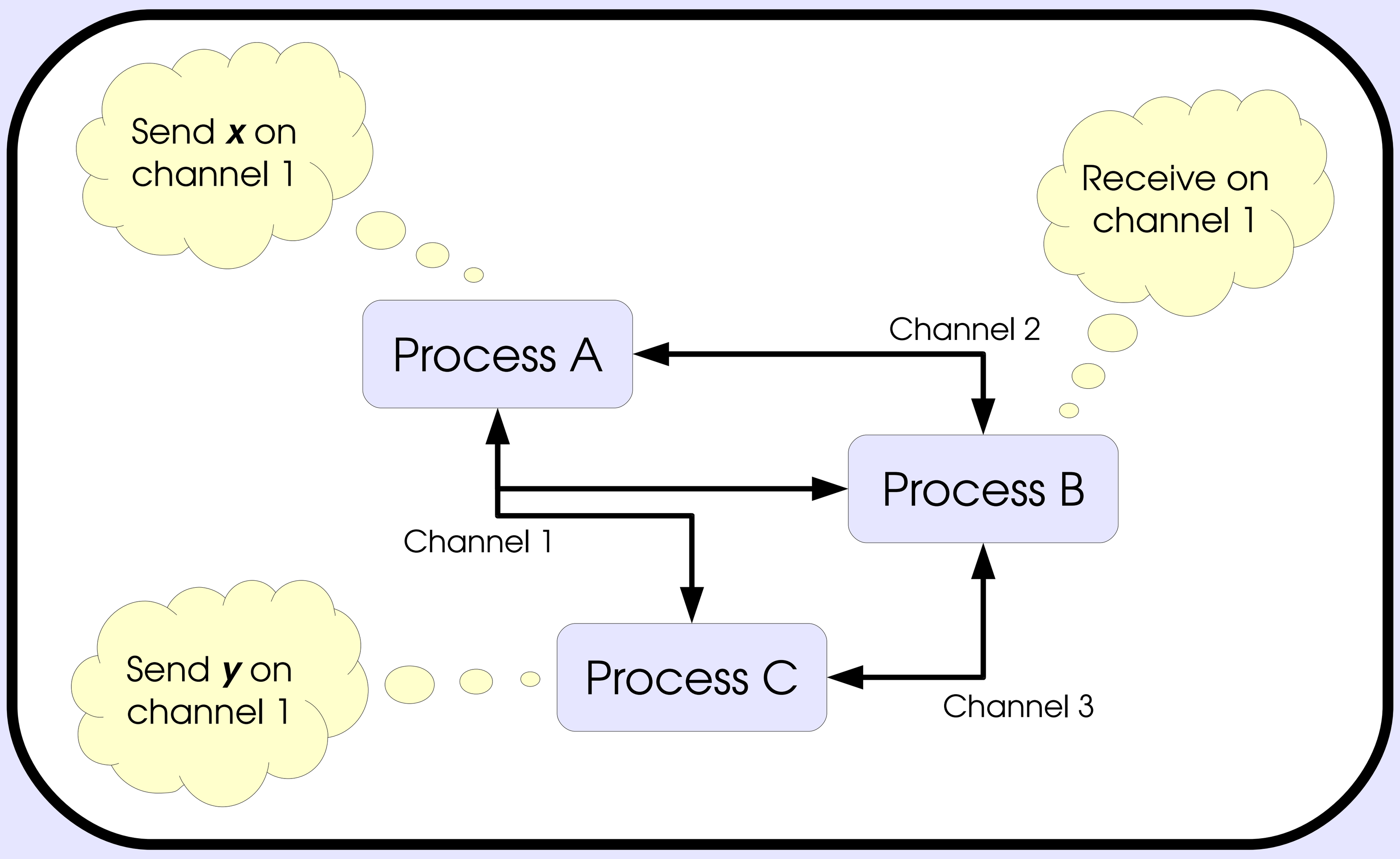

Figure 2: Message-passing concurrency

Challenges

What makes concurrency useful also makes it hard: the ability of one process to influence computation in another.

Influence is sometimes good (***communication***), sometimes bad (***interference***).

Important questions:
- How can you ***distinguish*** communication from interference?
- Does the ***mechanism*** (shared-state, message-passing) matter?
- How do you give a ***specification*** for a program if you don't know its environment?
- How can you show, ***modularly***, that a concurrent system satisfies its specification?

Goals

We want to state and check ***policies***

e.g.	if	messages along channel 1	come in sorted order
	then	messages along channel 3	leave in sorted order

This calls for a ***temporal logic*** where formulas are policies and models are processes.

A system is made up of many processes, so the logic should be ***compositional***:

$$\text{if } P \models \varphi \text{ and } Q \models \psi \text{ then } P|Q \models \varphi \otimes \psi$$

Idea

To manage interference, we introduce ***permissions*** into the logic:

- each process has send, receive permission on its channels
- permission is in $[0, 1]$
  - 0 means can't send/receive
  - 1 means no other process can send/receive on channel
  - otherwise, can send/receive, but so can other processes
- permissions for a channel must globally add up to 1
- processes exchange permissions as they communicate

By ***owning*** all the permission for a channel, a process can ensure no interference is possible.

Conclusions & Status

Our logic allows local reasoning about processes, regardless of their eventual environment:

- based on principles from ***separation logic***
- new application of ***fractional permissions*** – mobile, message-passing programs
- fully ***compositional*** – prove system correct by proving its components correct

We are close to proving the logic sound for the π-calculus.
We hope to apply it to the join-calculus/Microsoft's Polyphonic C#.