AARHUS UNIVERSITY

DEPARTMENT OF ENGINEERING

Test of Distributed Systems
Lecture 1

Introduction:

Course outline

Course plan

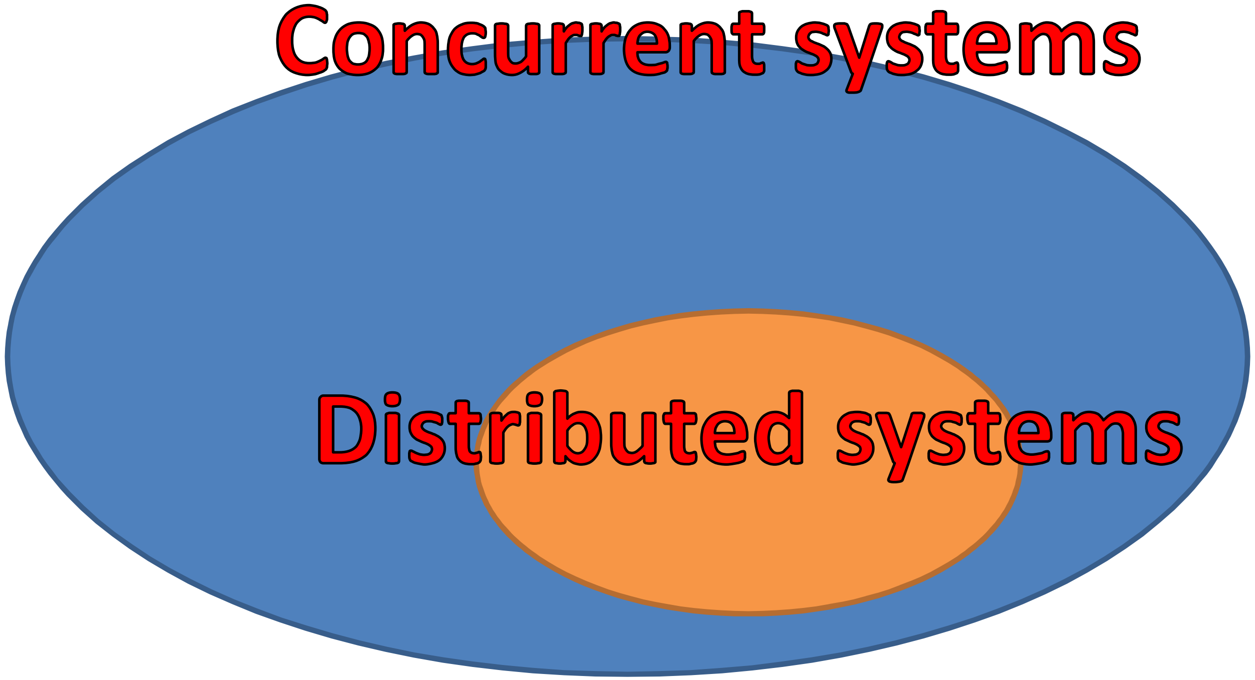
Starting with Spin & BACI

Today’s lecture

- Course outline
  - – Introduction
  - – Test, distributed system
- Starting with Spin & BACI – getting the tools installed
- Next time

Course outline

- We'll work with concurrent systems and distributed systems



Abstract concurrency - definition

- A concurrent system consists of a (finite) set of sequential processes. Each process executes a finite set of *atomic statements*. The concurrent system proceeds by executing a sequence of the atomic statements by *arbitrarily interleaving* atomic statements from the processes.
- Each process maintains a *control pointer* that indicates the next statement to execute by that process.

Example – 2 processes q & r

- p = { stmt1;stmt2 }, q = { stmt1;stmt2 }

p1→q1→p2→q2,

p1→q1→q2→p2,

p1→p2→q1→q2,

q1→p1→q2→p2,

q1→p1→p2→q2,

q1→q2→p1→p2.

Possible scenarios

Note:

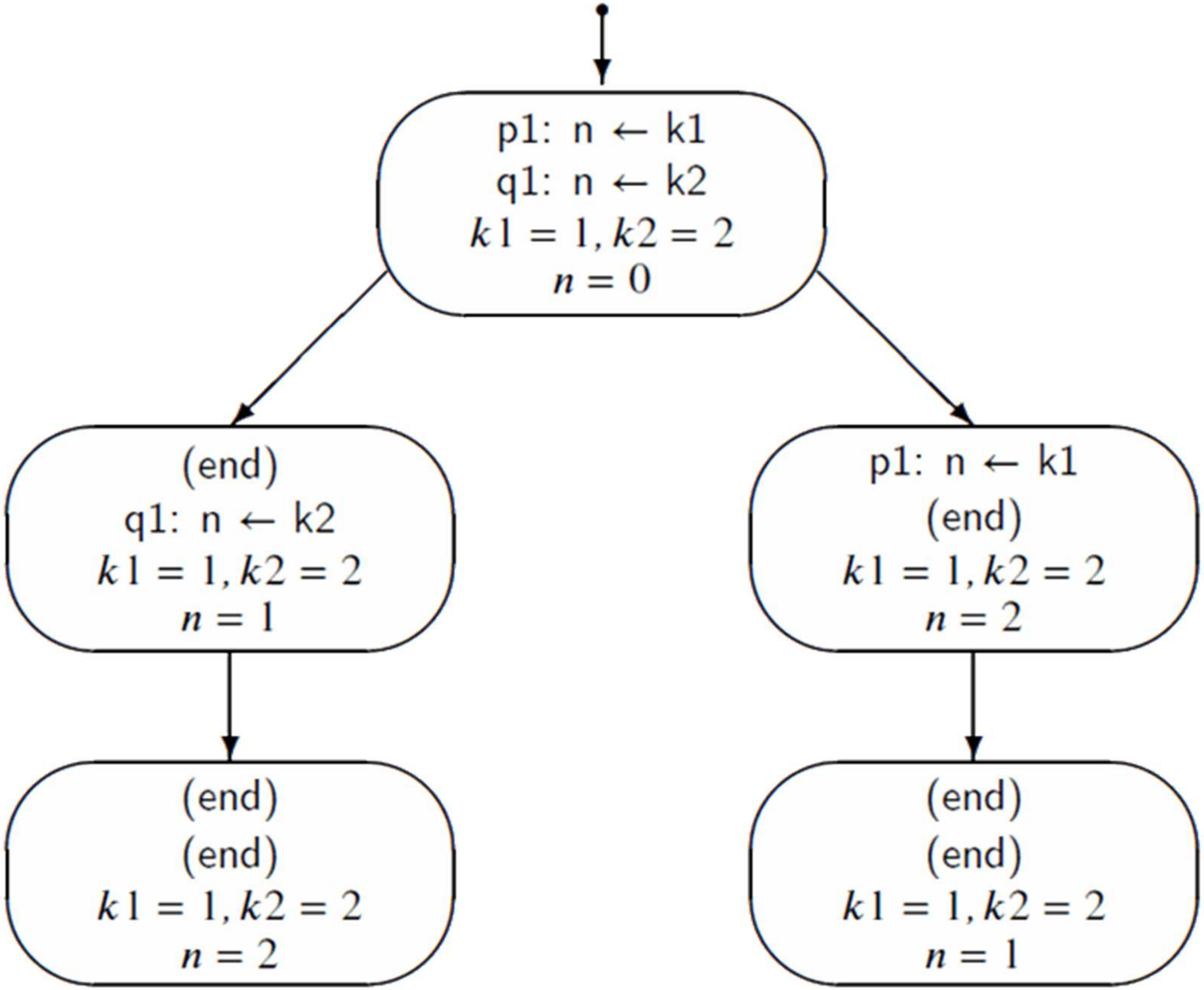
p2→p1→q1→q2

is *not* a valid scenario because it violates sequential execution of p1

Atomic statements and state

Algorithm: Trivial concurrent program	
integer n ← 0	
p	**q**
integer k1 ← 1 p1: n ← k1	integer k2 ← 2 q1: n ← k2

States, statements & transitions



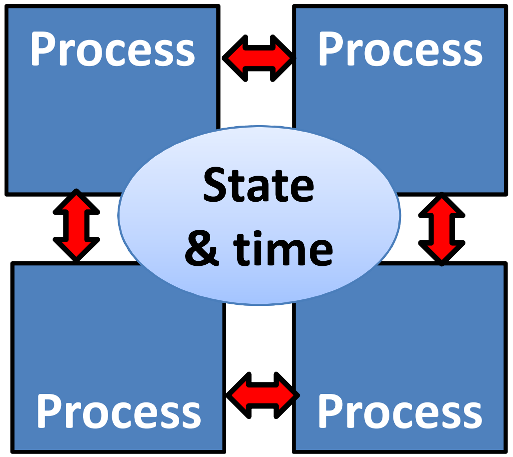
Non-determinism

- The arbitrary interleaving of process statements models the non-determinism of concurrent systems (we usually cannot predict or re-construct the order of the statements in relation to each other across processes).
- 2 statements from any 2 processes may or may not execute truly parallel. If they do, they can’t influence each other and we can arbitrarily choose one of 2 interleavings. If they don’t, some interleavings may be valid and some not.

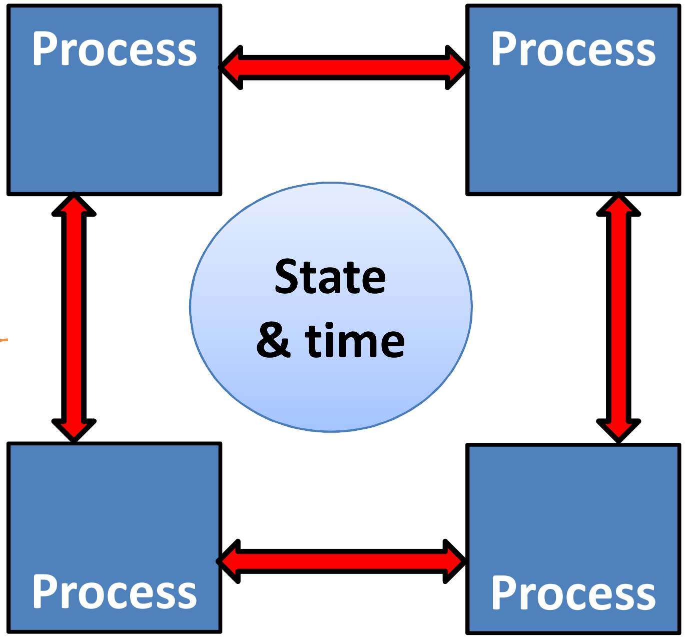
Non-determinism

- So some non-determinism results from the notion of concurrency. We cannot predict the exact order of individual statements, and hence not predict how the state of a concurrent program evolves. Different interleavings may result in the same final state
- But as long as all processes share the same time, we can at least observe the system's collective state at any given time.

From CoSy to DiSy

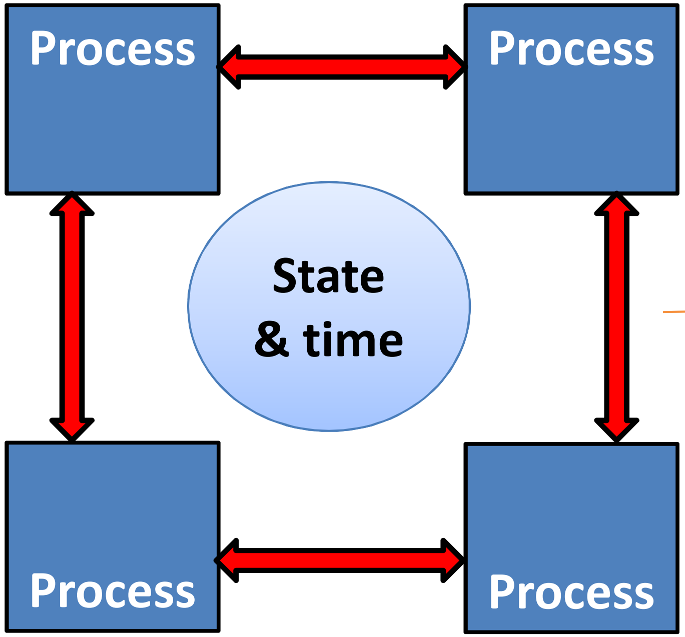


Shared time
Observable global state



No shared time
Global state not observable

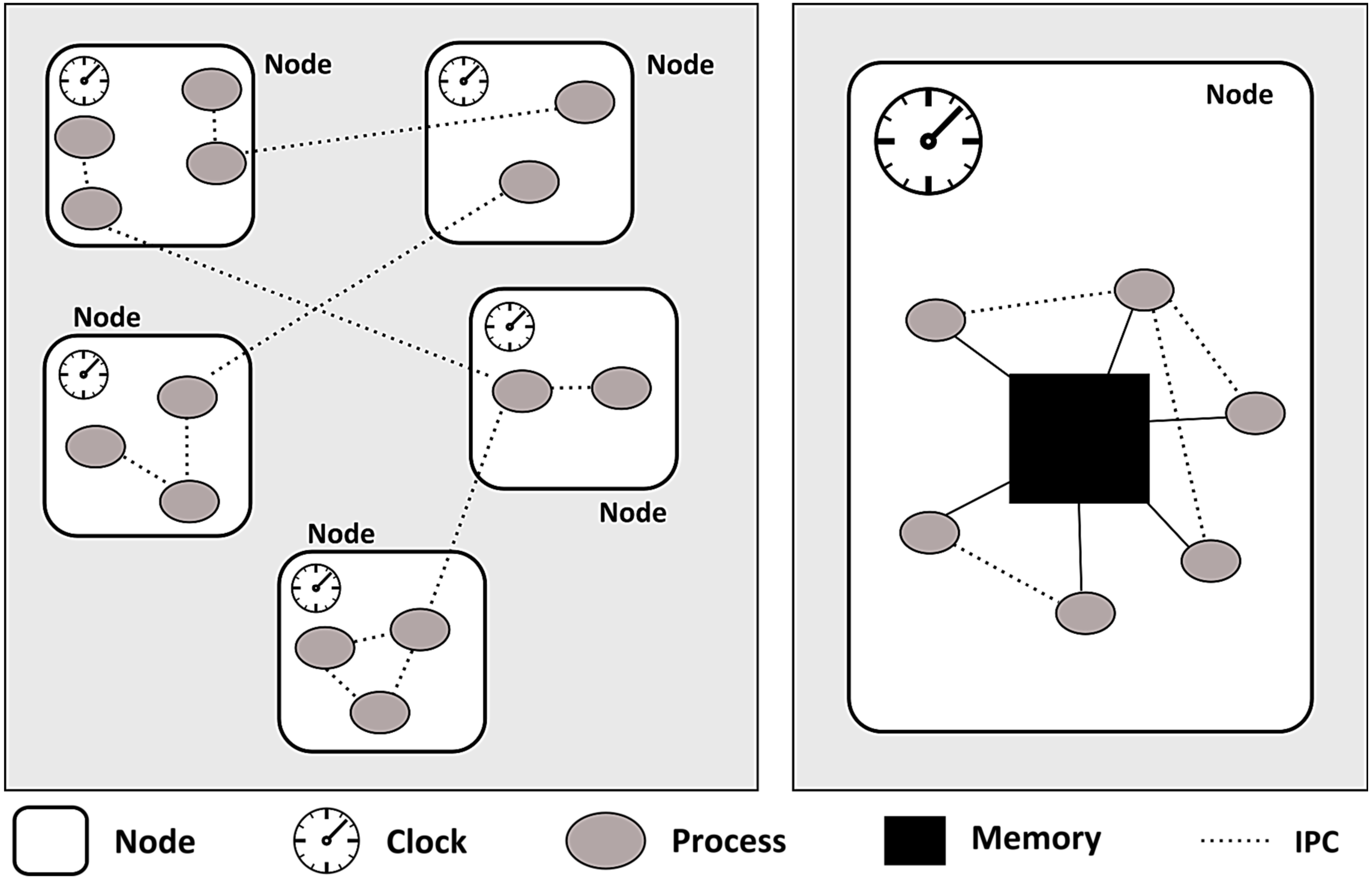
Channels



In distributed systems, the characteristics of the communication channels starts to play a major role:

- Delays
- Reordering
- Lossy or not
- Etc.

Course outline – our model of a distributed system



Non-determinism

- When we loose a shared, common time across processes, another type of non-determinism, or perhaps relativity, occurs
- We cannot observe the system’s global state!
- We can only observe the system through message exchanges, and by the time we have a status report from each process, their states may have changed
- The best we can do, is to construct images of global state that *may* have existed

Course contents

- The materials – let’s have a look
- The book
  “Principles of the Spin Model Checker”
- The tool
  - (j)Spin & (j)BACI

Coarse course outline

- Introduction & install fest (today)
- 1st (SHA) – Concurrency (1 ½ weeks)
  - Basic concepts
  - Modelchecking, Spin
  - Implementation/test, BACI
- 2nd part (SKR) – Distributed system (3 weeks)
  - Theory (consistent states, causality)
  - Case study of distributed algorithm
  - Use Spin/BACI to study, extend & verify modified algorithm
- 3rd part (SHA) Concurrency contd.

What about the test part?

- Lots of talk about concurrency & distribution, but what about the test thing?
- We apply a broad view of test
  - Modelchecking w. Spin = test/verification of design
  - Prototyping (in BACI) = method for developing test cases
- Why this broad view?
  - Test of CS/DS is hard, requires understanding of the issues involved
  - There is no generally applied/accepted tools or methods used in industry

Starting with Spin & jBACI

- Material:
  - The article (“A Primer om Model Checking”)
  - The short manual on JBACI and BACI-C (C--)
- Demo
- Installation
- A little exercise

Starting with jBACI

- What does it do?
  - Compiles and runs concurrent programs written in a subset of C++ called baci-c or C—
  - Programs can simply be run in their entirety
  - But the interesting feature is the ability to execute concurrent programs step-by-step (atomic statement by atomic statement) across processes – we get to choose and study the effect of a specified interleaving
- Demo: add.cm

Starting with Spin
- what does it do

- Several things
- The four “modes” of Spin
  - Random simulation (example program run)
  - Interactive simulation (guided by user)
  - Verification (full state space exploration)
  - Guided simulation (using trail from verification)
- Verification of the full state space is the interesting part (and more or less the only one)

Starting with Spin
- how does it work

- Won’t go into small nitty-gritty details
- The most important aspect from a user perspective (at least initially) is understanding what ProMeLa is and how to use it
- **Pro**cess **Me**ta **La**nguage
  - For modelling systems/protocols
  - Not for hard-core programming
  - Abstraction is the key

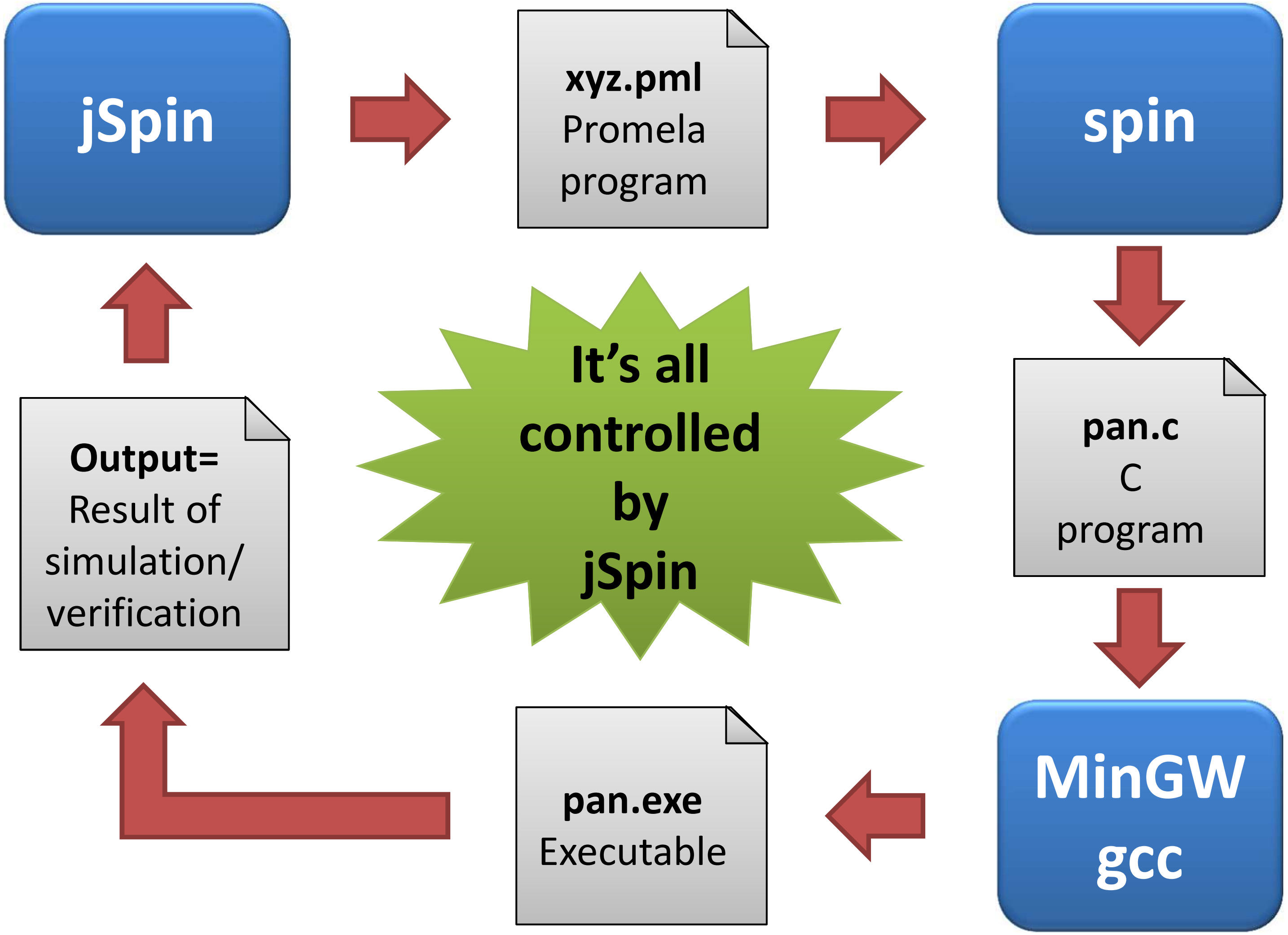
Starting with Spin - installation

- It’s a small footprint installation
- We’ll use the jSpin frontend
- Entire directory can simply be deleted for un-installing jSpin/Spin
- Java is assumed
- Use the installation guide
- Let’s go …

Starting with Spin - demo

- Running Spin via jSpin
- Looking at a few small programs to see ProMeLa at work

(j)Spin – behind the scenes



jSpin

xyz.pml Promela program

spin

pan.c C program

MinGW gcc

pan.exe Executable

Output= Result of simulation/ verification

It's all controlled by jSpin

Starting with Spin 6 JBACI - a few small exercises

- Put on the hard hat and get to work …
- First do the install boogie
- Then use

Starting with Spin
- a few small exercises

- Exercise 1 (write this in both Spin & BACI)
  - Make a small program with 3 processes.
  - ProcessA should start ProcessB and ProcessC
  - When started, processB should increment a global counter from 0 to 10, and then terminate
  - When started, ProcessC should wait for the global counter to reach 10, and then count it down to 0 before terminating
  - ProcessA should wait for ProcessB and ProcessC to terminate, write the value of the global counter and then terminate