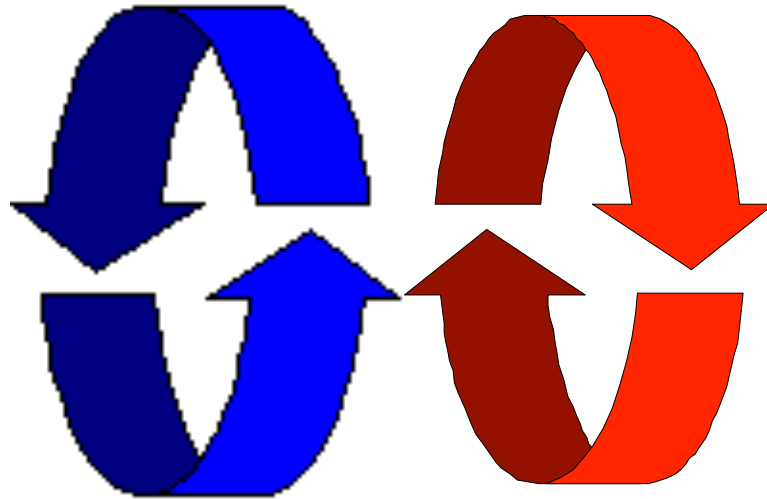


# Concurrency

---

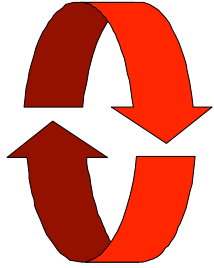
## *Concepts, Models and Programs*



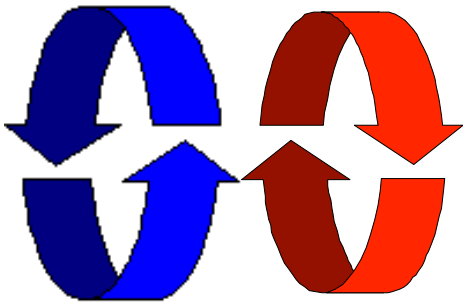
**Jeff Kramer**

# What is a Concurrent Program?

---



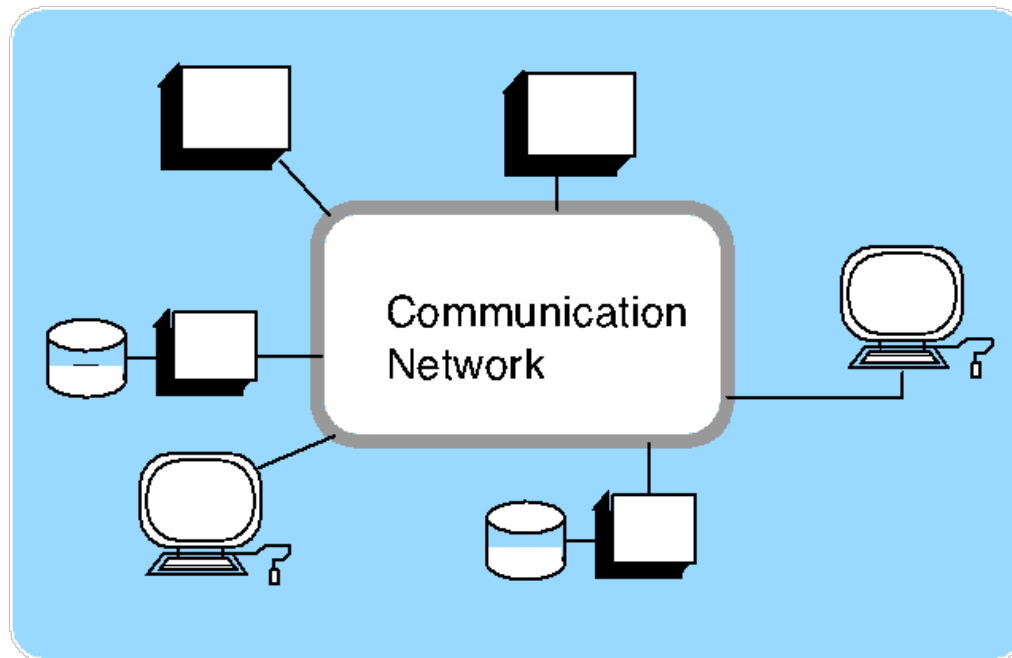
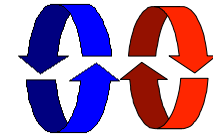
A **sequential** program has a single thread of control.



A **concurrent** program has multiple threads of control allowing it perform multiple computations in parallel and to control multiple external activities which occur at the same time.

# Concurrent and Distributed Software?

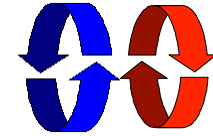
---



Interacting, concurrent software components of a system:

single machine ->  
*shared memory*  
*interactions*

multiple machines ->  
*network interactions*



# Why Concurrent Programming?

---

- ◆ Performance gain from multiprocessing hardware
  - eg. fine grain parallelism on multicore hardware : low level memory models
  - eg. coarse grain parallelism for partitioned scientific calculations : processes
- ◆ Increased application throughput : **avoid polling (busy waiting)!**
  - eg. an I/O call need only block one thread
- ◆ Increased application responsiveness
  - eg. high priority thread for user requests.
- ◆ **More appropriate structure**
  - for programs which interact with the environment, control multiple activities and handle multiple events – coarse grain parallelism.

# Concurrency is widespread but error prone!

---

*A very simple example:*

- ◆ process 1:  $x := x + 1$  (x shared variable)
- ◆ process 2:  $x := x - 1$  (x shared variable)

Final result?

Single line instructions are generally **not** atomic.

Assuming read and write are atomic, the result depends on the order of read and write operations on x!

**We have a race condition!**

## Concurrency is widespread but error prone!

---

There are 4 atomic x-operations:

Process 1 reads x (R1), then writes to x (W1).

Process 2 reads x (R2), then writes to x (W2).

R1 must happen before W1 and R2 before W2, so these operations can be sequenced in 6 ways (x initially 0):

R1	R1	R1	R2	R2	R2
W1	R2	R2	R1	R1	W2
R2	W1	W2	W1	W2	R1
W2	W2	W1	W2	W1	W1
<hr/>					
0	-1	1	-1	1	0

We see that the final value of x is **-1, 0, or 1**. The program is thus **non-deterministic** : the result can vary from execution to execution.

# Concurrency is widespread but error prone!

---

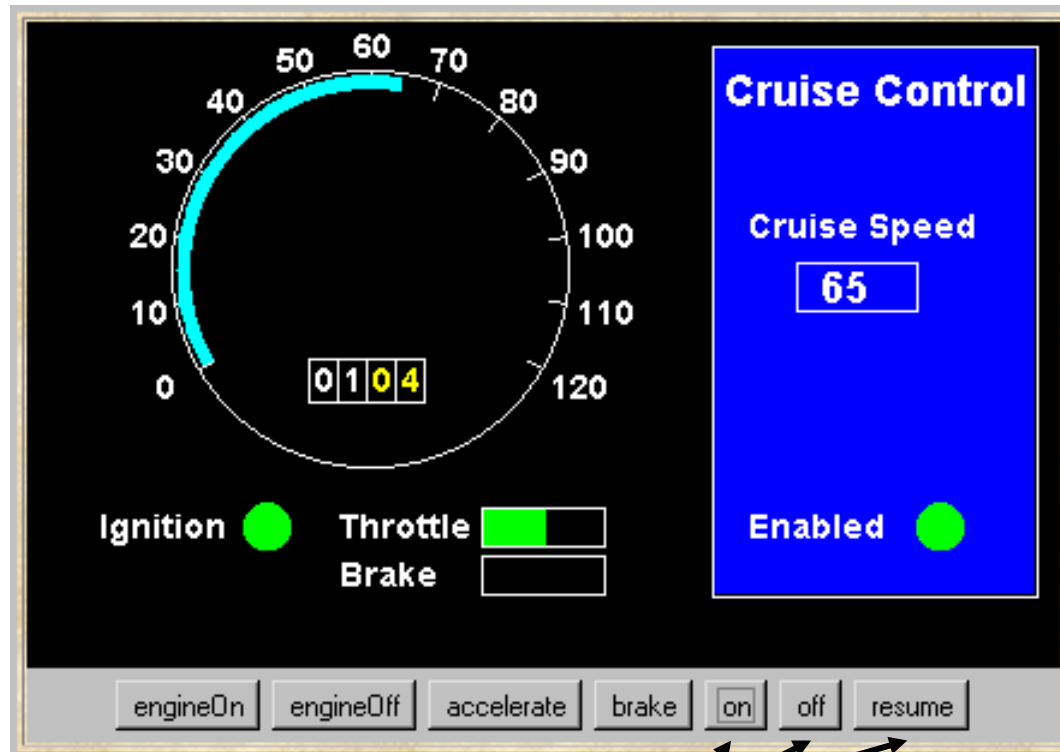
- ◆ Therac - 25 computerised radiation therapy machine

**Concurrent programming errors contributed to accidents causing deaths and serious injuries.**

- ◆ Mars Rover

**Problems with interaction between concurrent tasks caused periodic software resets reducing availability for exploration.**

## a Cruise Control System



When the car ignition is switched on and the **on** button is pressed, the current speed is recorded and the system is enabled: *it maintains the speed of the car at the recorded setting.*

Pressing the brake, accelerator or **off** button disables the system. Pressing **resume** re-enables the system.

- ◆ *Is the system safe?*
- ◆ *Would testing be sufficient to discover all errors?*



# Models for concurrent programming

---

*Engineering is based on the use of simpler, **abstract** models for experimentation, reasoning and exhaustive analysis.*



## Abstraction? definitions ...

---

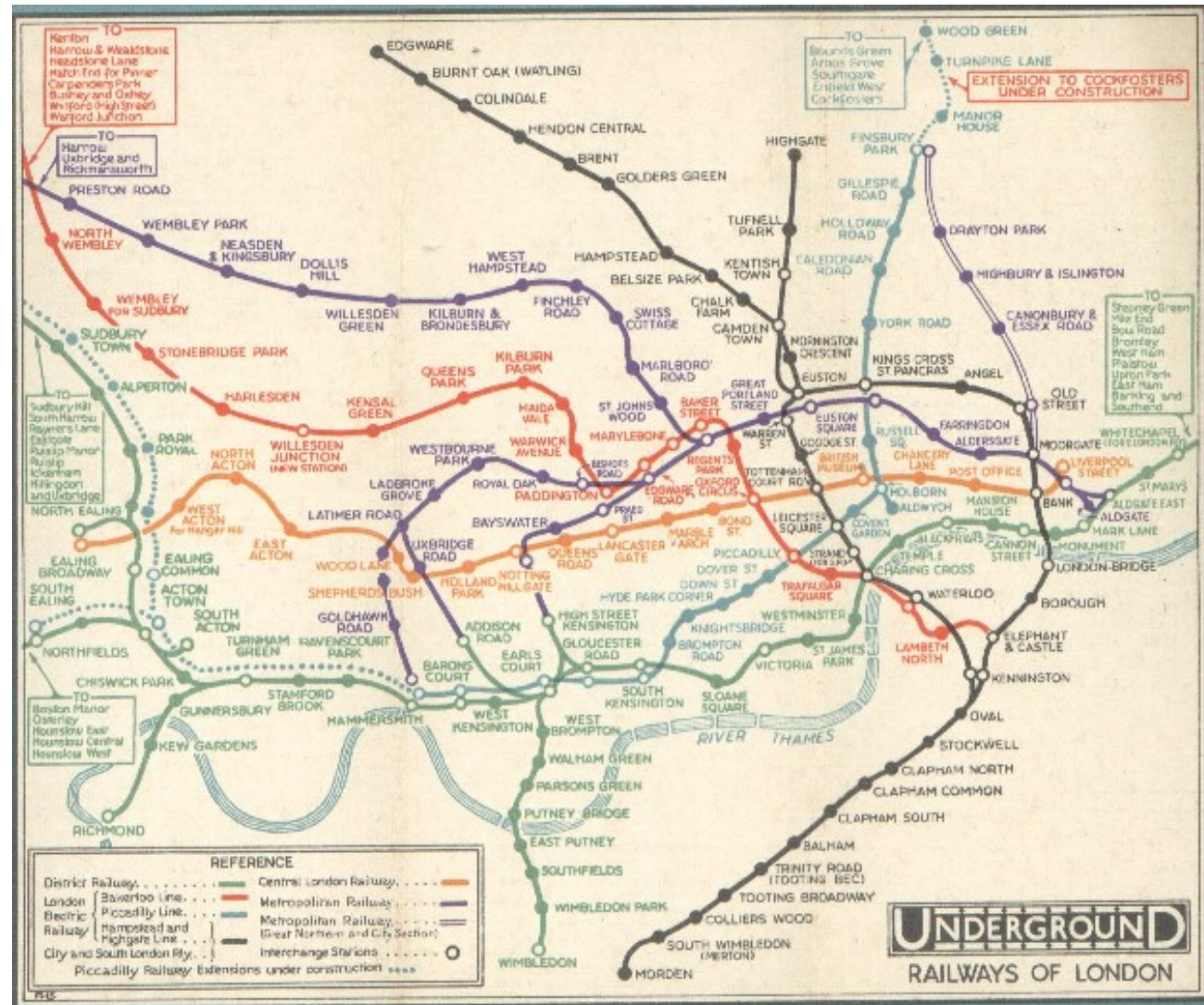
- the act of withdrawing or removing something
- the act or process of leaving out of consideration one or more properties of a complex object so as to attend to others

*Remove detail (simplify) and  
focus (selection based on purpose)*

- a general concept formed by extracting common features from specific examples
- the process of formulating general concepts by abstracting common properties of instances

*Generalisation (core or essence)*

# 1930 – London Underground map



**“Fit for purpose?”**

**Purpose:**  
relationship  
between  
stations and the  
interchanges,  
**not** actual  
distances.



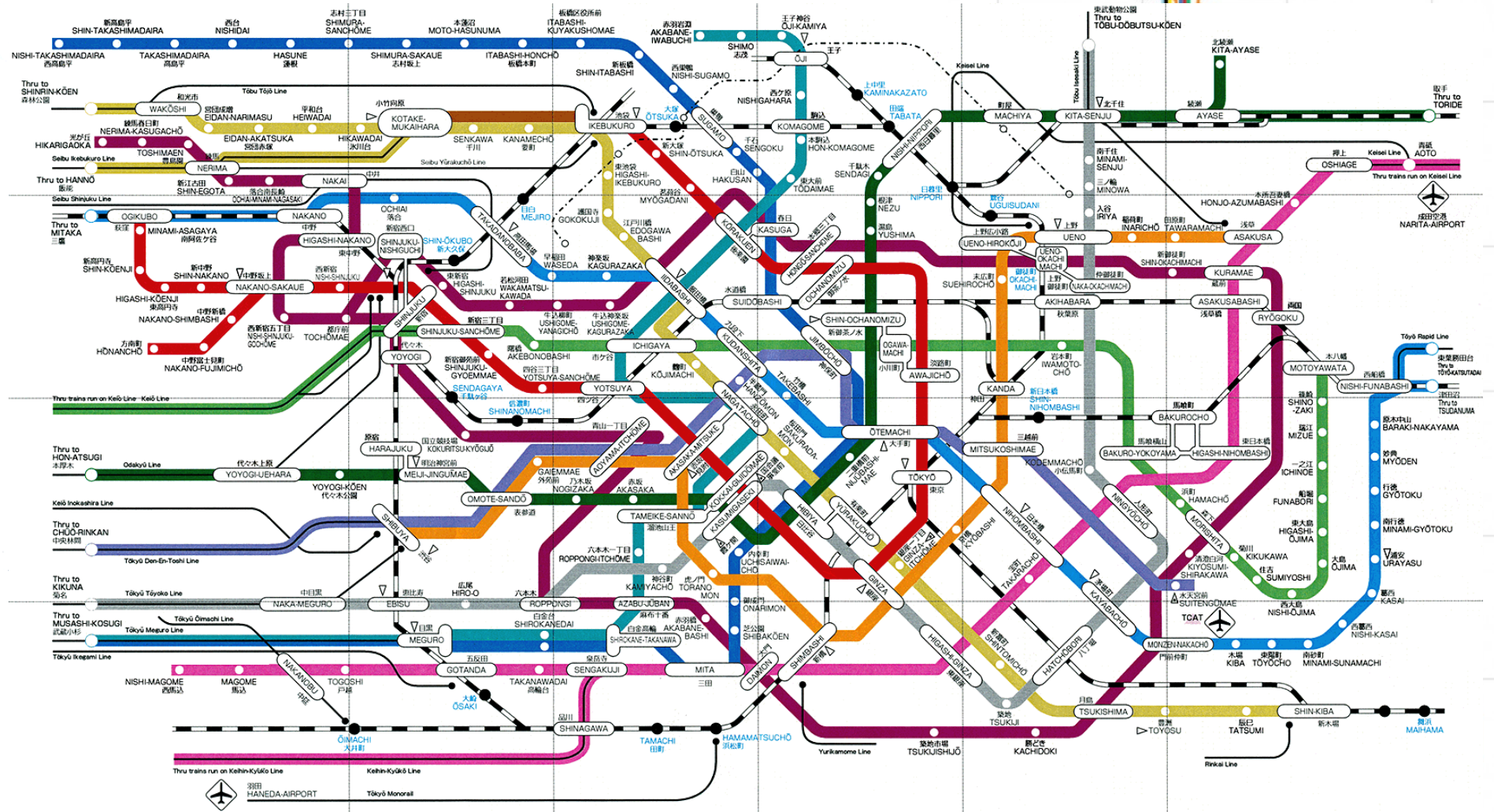


# 2001 – Fit for purpose (“mind the gap...”)





# 2001 – Fit for purpose?!



## 2015 Concurency: introduction

## Why is abstraction important in **Software Engineering**?

---

*Software is abstract!*

“Once you realize that computing is all about constructing, manipulating, and reasoning about abstractions, it becomes clear that an important prerequisite for writing (good) computer programs is the ability to handle abstractions in a precise manner.”

**Keith Devlin CACM Sept.2003**

*Perhaps abstraction is the key to computing .... ?*

**CACM April 2007**

# Models for concurrent programming

---

*Engineering is based on the use of simpler, **abstract** models for experimentation, reasoning and exhaustive analysis.*

**Abstraction** is fundamental to Engineering in general, and to Software Engineering in particular !





## Models and Model Checking

---

A **model** is an abstract, simplified representation of the real world.

Engineers use models to gain confidence in the adequacy and validity of a proposed design:

- ◆ focus on an aspect of interest - concurrency
- ◆ model animation to visualise a behaviour
- ◆ automated **model checking of properties** (safety & progress)

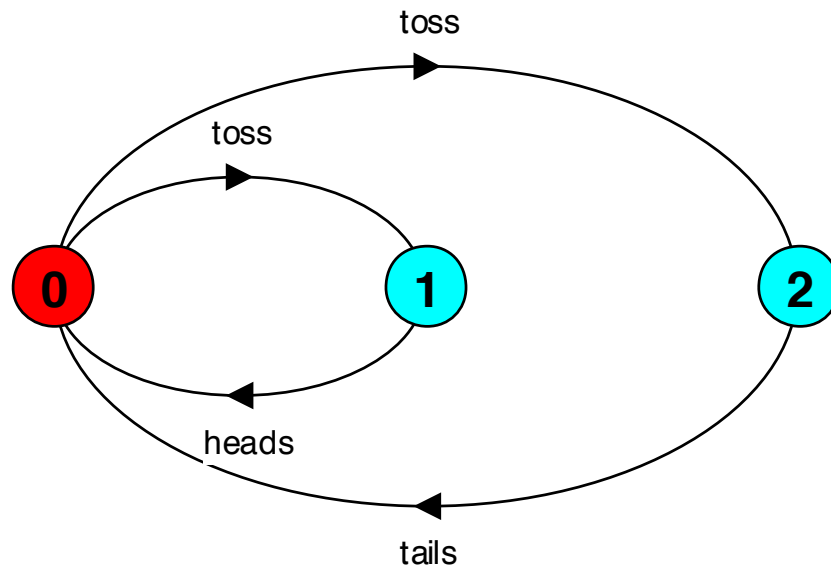
Models are needed to

- ◆ experiment, test, and check a design before it is implemented
  - ◆ airplane software before test flight
  - ◆ net bank services before customer use
  - ◆ medical sensor system before patient use
- ◆ may model an environment, hardware units, partly unknown parts, third party software, ...

## Models and Model Checking

---

Models are described using state machines, known as Labelled Transition Systems **LTS**. These are described textually in a Process Algebra as finite state processes (**FSP**) and displayed and analysed by the **LTSA** model checking analysis tool.



```
COIN = (toss->HEADS  
        | toss->TAILS) ,  
HEADS= (heads->COIN) ,  
TAILS= (tails->COIN) .
```

# Finite State Machines and Model Checking

---

## Turing awards related to Model Checking

### ◆ (2007) Edmund M. Clarke, E. Allen Emerson, Joseph Sifakis:

- “for their role in developing Model-Checking into a highly effective verification technology that is widely adopted in the hardware and software industries.”

### ◆ (1996) Amir Pnueli:

- “For seminal work introducing temporal logic into computing science and for outstanding contributions to program and system verification.”

### ◆ (1986) John E Hopcroft, “Bob” Tarjan:

- foundation of formal languages for state machines

### ◆ (1976) Michael O. Rabin, Dana S. Scott:

- Finite Automata and Their Decision Problem, which introduced the idea of nondeterministic machines, which has proved to be an enormously valuable concept.

# Process Algebra

---

## Turing awards related to Process Algebra

### ◆ (1991) “Robin” Milner:

For three distinct and complete achievements:

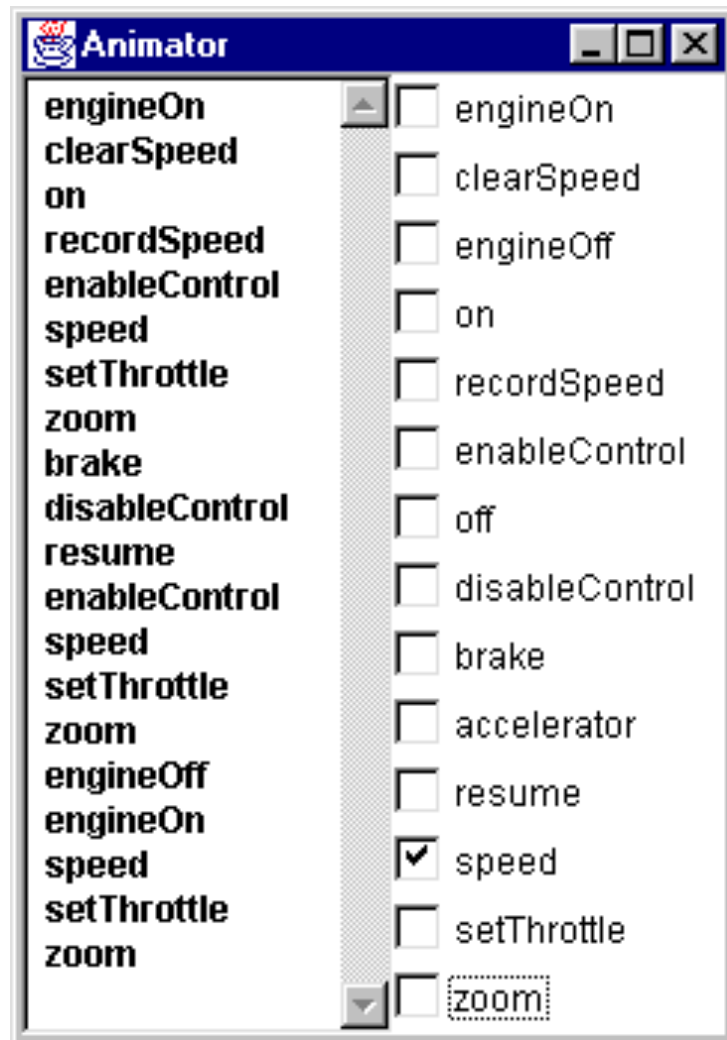
- LCF the mechanization of Scott’s Logic of Computable Functions, for machine assisted proof construction;
- ML the first language to include polymorphic type inference together with a type-safe exception-handling mechanism;
- Process Algebra: CCS (Calculus of Communicating Systems) a general theory of concurrency.

### ◆ (1980) “Tony” Hoare.

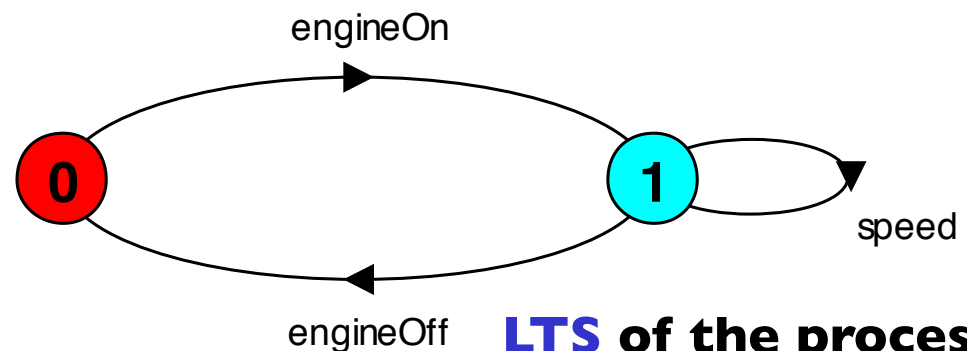
For his fundamental contributions to the definition and design of programming languages..”

- Process Algebra: the language and theory around CSP (Communicating Sequential Processes)

# modelling the Cruise Control System



**LTSA Animator to step through system actions and events.**



**LTS of the process that monitors speed.**

Later chapters will explain how to construct models such as this so as to perform animation and verification.

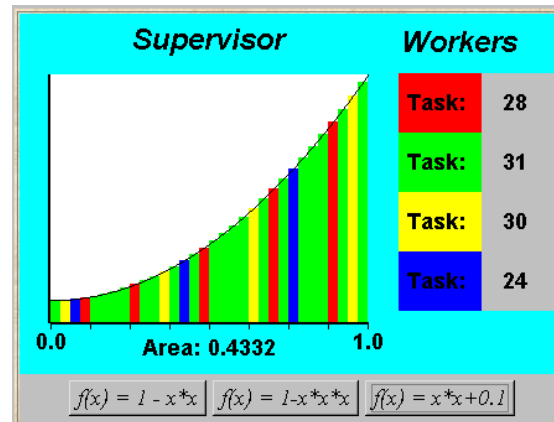
## programming practice in Java

---

Java is

- ♦ widely available, generally accepted and portable
- ♦ provides sound set of concurrency features

Hence Java is used for all the illustrative examples, the demonstrations and the exercises. Later chapters will explain how to model, check and construct Java programs such as the Cruise Control System and others ...



## course objectives

---

This course is intended to provide a sound understanding of the basic **concepts**, **models** and **practice** involved in designing concurrent software.

The emphasis on principles and **concepts** provides a thorough understanding of the issues and the solutions. **Modelling** provides insight into concurrent behavior and aids reasoning about particular designs. Concurrent programming in **Java** provides the programming **practice** and experience.

## Learning outcomes...

---

After completing this course, you will know

- ◆ how to model, analyze, and program concurrent object-oriented systems.
- ◆ the most important concepts and techniques for concurrent programming.
- ◆ what are the problems which arise in concurrent programming.
- ◆ what techniques you can use to solve these problems.



## Book

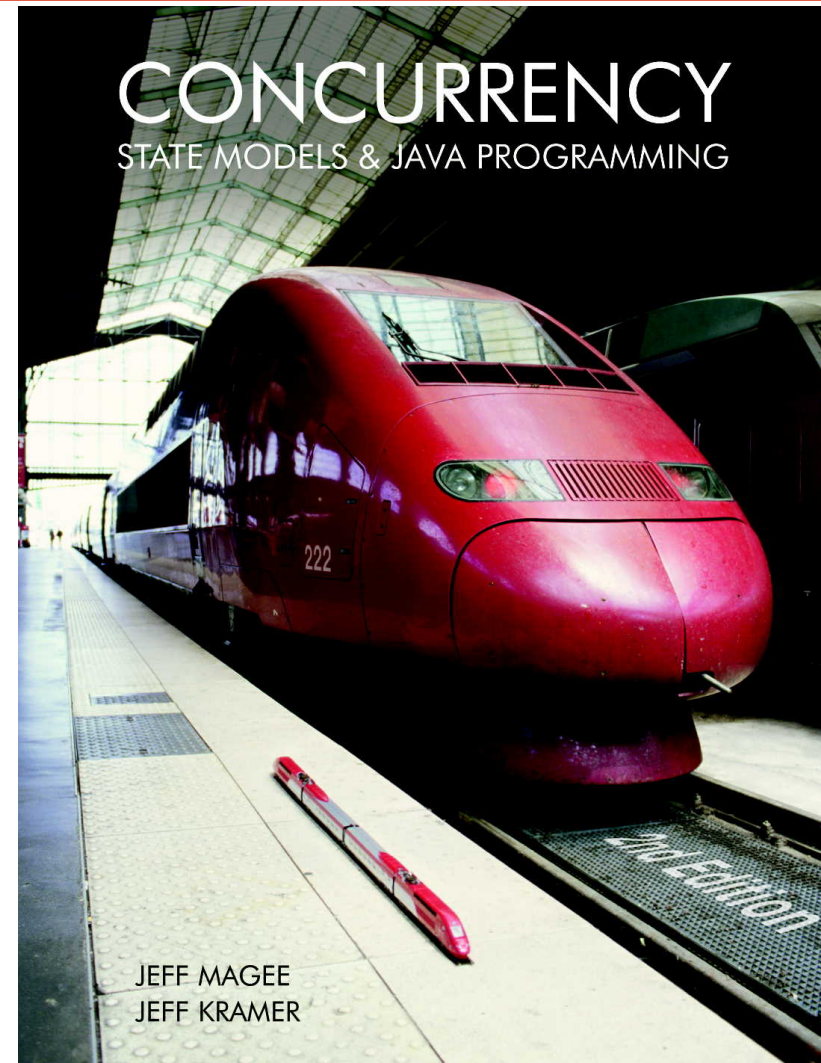
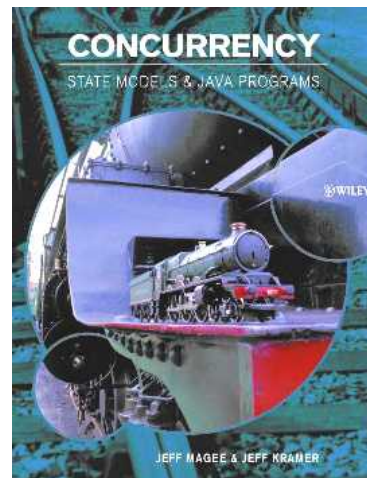
---

# Concurrency: State Models & Java Programs, 2<sup>nd</sup> Edition

Jeff Magee &  
Jeff Kramer

WILEY

1<sup>st</sup>  
edition



# Course Outline

---

- 2. Processes and Threads**
- 3. Concurrent Execution**
- 4. Shared Objects & Interference**
- 5. Monitors & Condition Synchronization**
- 6. Deadlock**
- 7. Safety and Liveness Properties**
- 8. Model-based Design**

*The main basic*  
**Concepts**  
**Models**  
**Practice**

*\* Advanced topics ...*

9. Dynamic systems

10. Message Passing

11. Concurrent Software Architectures

12. Timed Systems

13. Program Verification

14. Logical Properties

**\*\* Advanced concurrent programming ... mainly Java SE 8, some C**

## Web based course material

---

<http://www-dse.doc.ic.ac.uk/concurrency/>

- ◆ Java examples and demonstration programs
- ◆ State models for the examples
- ◆ Labelled Transition System Analyser (**LTSA**) for modelling concurrency, model animation and model property checking.

## Summary

---

### ◆ Concepts

- **we adopt a model-based approach for the design, analysis and construction of concurrent programs**

### ◆ Models

- **we use finite state models to represent concurrent behaviour.**

### ◆ Practice

- **we use Java for constructing concurrent programs.**

*Examples are used to illustrate the concepts, models and demonstration programs.*