Embedded System Design and Synthesis

Robert Dick

http://robertdick.org/esds/
Office: EECS 2417-E
Department of Electrical Engineering and Computer Science
University of Michigan

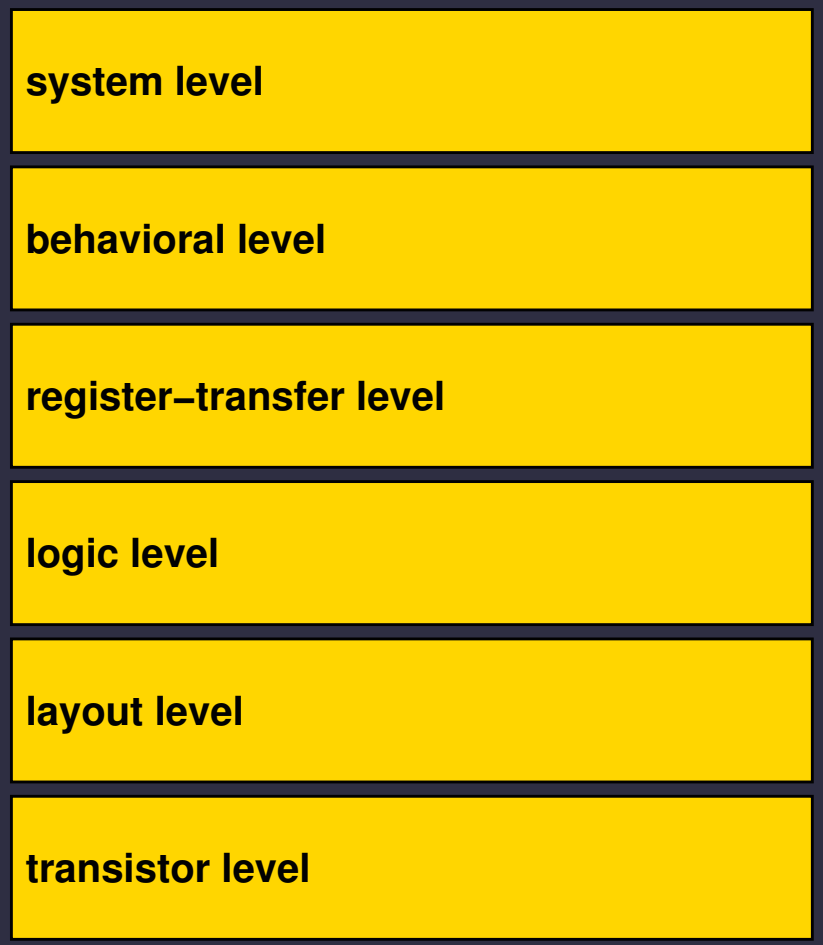
Today's plan

- Specification and modeling languages.
- Comments on mini-project selection.
- Example specification project.

Outline

Section outline

Short and fat vs. tall and thin



Available resources – System-level

- General-purpose SW processors
- Digital signal processors (DSPs)
- Application-specific integrated circuits
- Dynamically reconfigurable hardware
  - E.g., field-programmable gate arrays (FPGAs)
- Busses
- Wireless communication channels
- Wires

Available resources – High-level

- Simple arithmetic/logic units
- Multiplexers
- Registers
- Wires

Specification language requirements

- Describe hardware (HW) and software (SW) requirements
- Specify constraints on design
- Indicate system-level building blocks
- To allow flexibility in synthesis, must be abstract
  - Differentiate HW from SW only when necessary
  - Concentrate on requirements, not implementation
  - Make few assumptions about platform

Section outline

1. Specification and modeling languages
   - Introduction
   - Software oriented design representations
   - Hardware oriented design representations
   - Graph based design representations
   - Resource descriptions

Software oriented design representations

- ANSI-C
- SystemC
- Other SW language-based

ANSI-C

Advantages

- Huge code base
- Many experienced programmers
- Efficient means of SW implementation
- Good compilers for many SW processors

Disadvantages

- Little implementation flexibility
  - Strongly SW oriented
  - Makes many assumptions about platform
- Poor support for fine-scale HW synchronization

SystemC

Advantages

- Support from big players
  - Synopsys, Cadence, ARM, Red Hat, Ericsson, Fujitsu, Infineon Technologies AG, Sony Corp., STMicroelectronics, and Texas Instruments
- Familiar for SW engineers

Disadvantages

- Extension of SW language
  - Not designed for HW from the start
- Compiler available for limited number of SW processors
  - New

Other SW language-based

- Numerous competitors
- Numerous languages
  - ANSI-C, C++, and Java are most popular starting points
- In the end, few can survive
- SystemC has broad support

Section outline

Hardware oriented design representations

- VHDL
- Verilog
- Esterel

VHDL

Advantages

- Supports abstract data types
- System-level modeling supported
- Better support for test harness design

Disadvantages

- Requires extensions to easily operate at the gate-level
- Difficult to learn
- Slow to code

Verilog

Advantages

- Easy to learn
- Easy for small designs

Disadvantages

- Not designed to handle large designs
- Not designed for system-level

Verilog vs. VHDL

- March 1995, Synopsys Users Group meeting
- Create a gate netlist for the fastest fully synchronous loadable 9-bit increment-by-3 decrement-by-5 up/down counter that generated even parity, carry and borrow
- 5 / 9 Verilog users completed
- 0 / 5 VHDL users competed

Does this mean that Verilog is better?
Maybe, but maybe it only means that Verilog is easier to use for simple designs.

Esterel

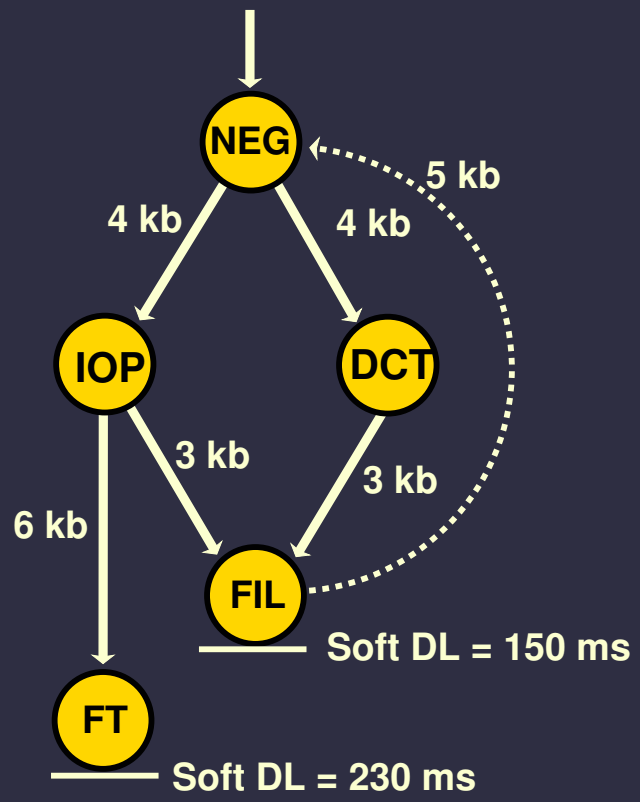
- Easily allows synchronization among parallel tasks
- Works above RTL
  - Doesn't require explicit enumeration of all states and transitions
- Recently extended for specifying datapaths and flexible clocking schemes
- Amenable to theorem proving
- Translation to RTL or C possible
- Commercialized by Esterel Technologies

Section outline

Graph based design representations

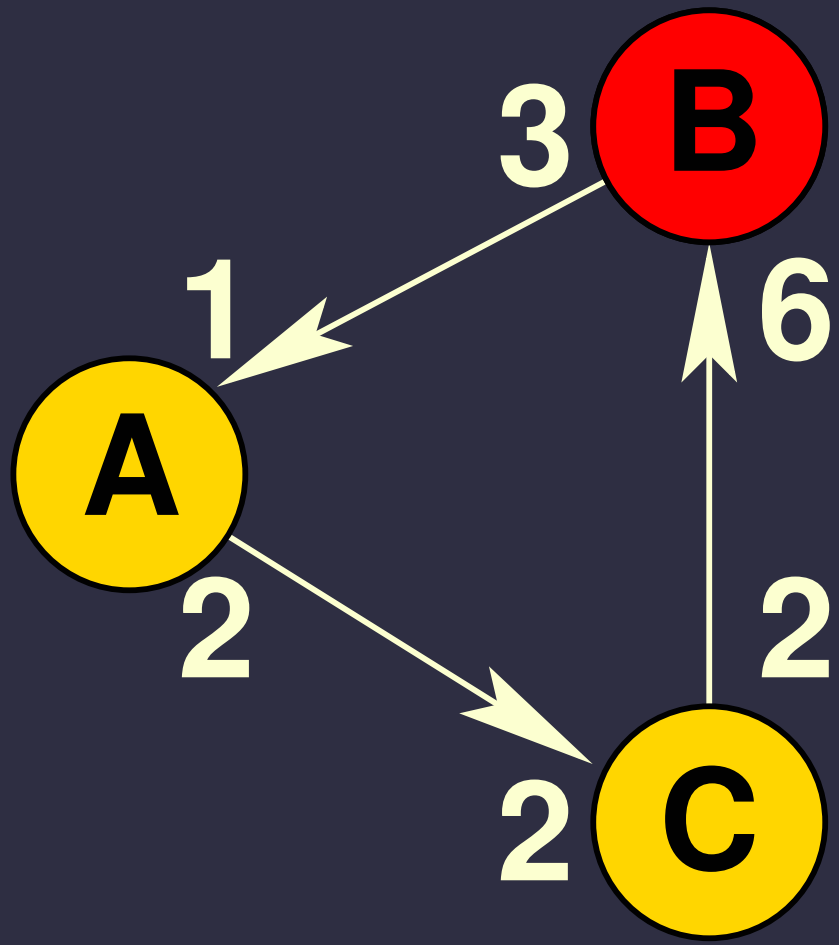
- Dataflow graph (DFG)
- Synchronous dataflow graph (SDFG)
- Control flow graph (CFG)
- Control dataflow graph (CDFG)
- Finite state machine (FSM)
- Petri net
- Periodic vs. aperiodic
- Real-time vs. best effort
- Discrete vs. continuous timing
- Example from research

Dataflow graph (DFG)

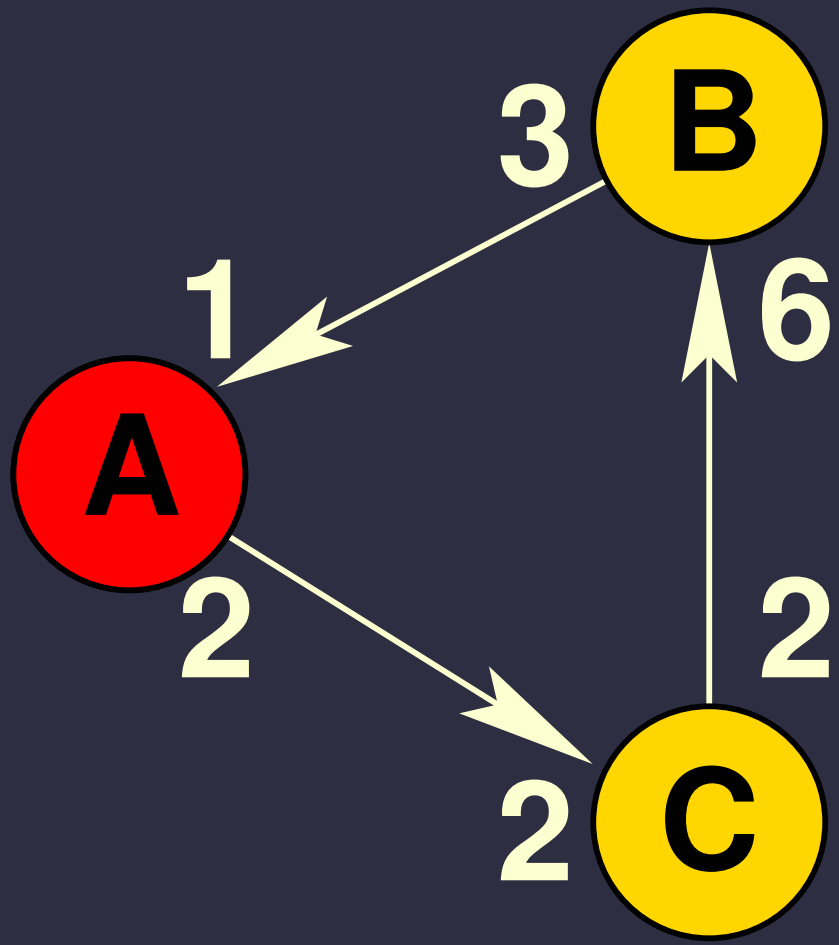


- Nodes are tasks
- Edges are data dependencies
- Edges have communication quantities
- Used for digital signal processing (DSP)
- Often acyclic when real-time
- Can be cyclic when best-effort

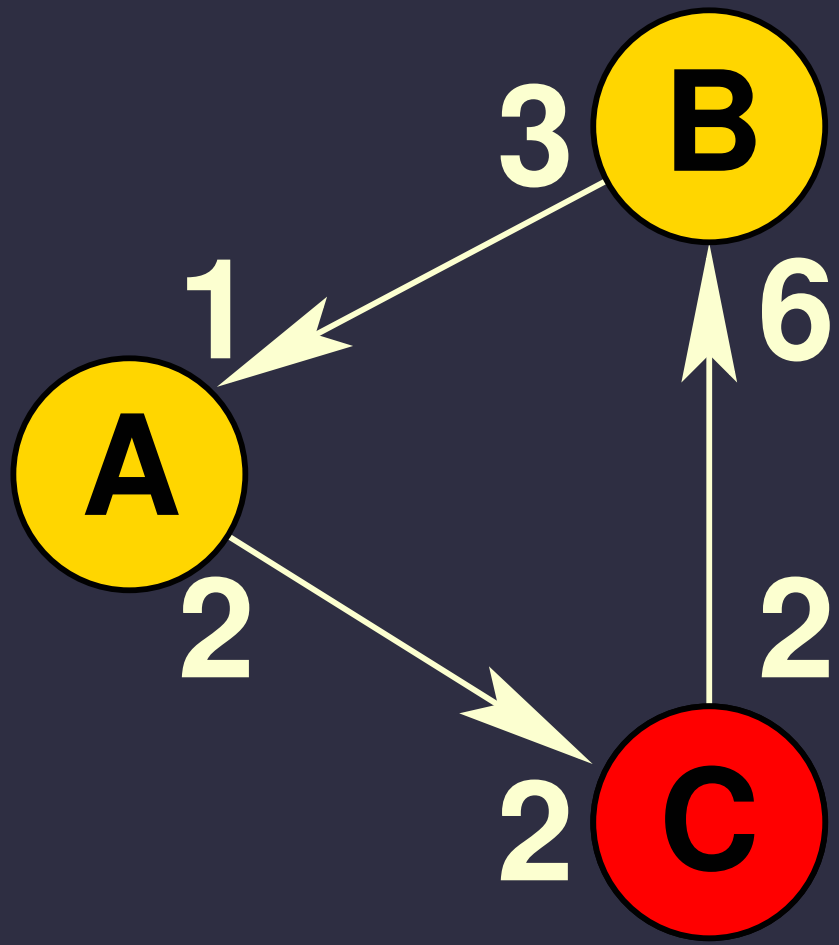
Synchronous dataflow graph (SDFG)



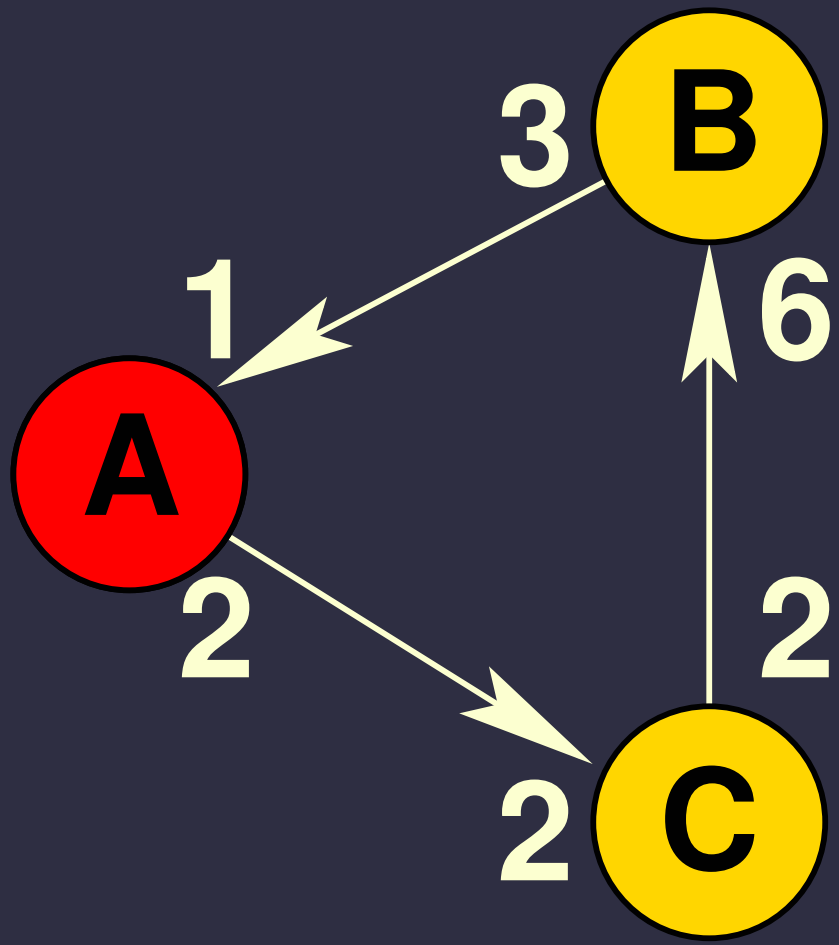
Synchronous dataflow graph (SDFG)



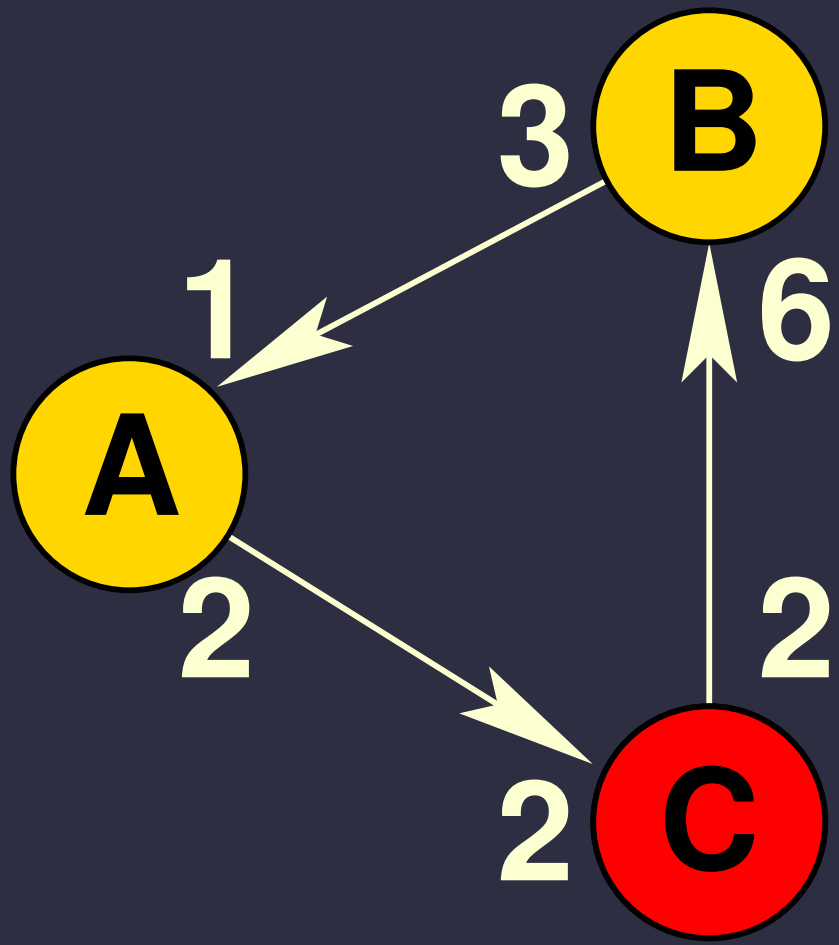
Synchronous dataflow graph (SDFG)



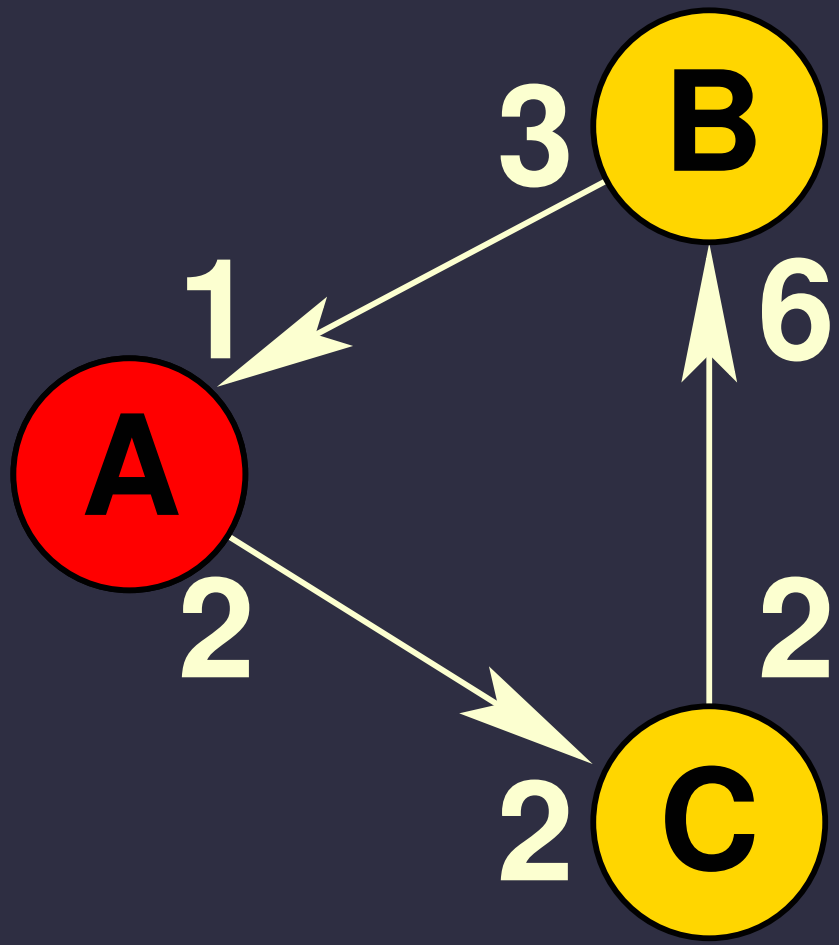
Synchronous dataflow graph (SDFG)



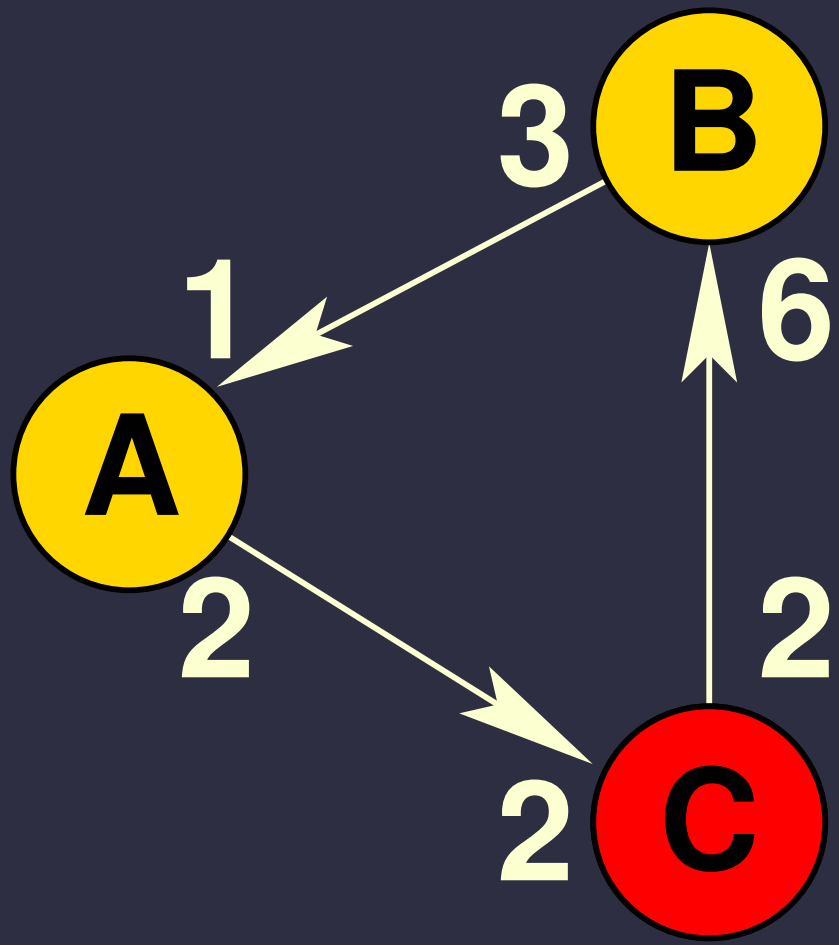
Synchronous dataflow graph (SDFG)



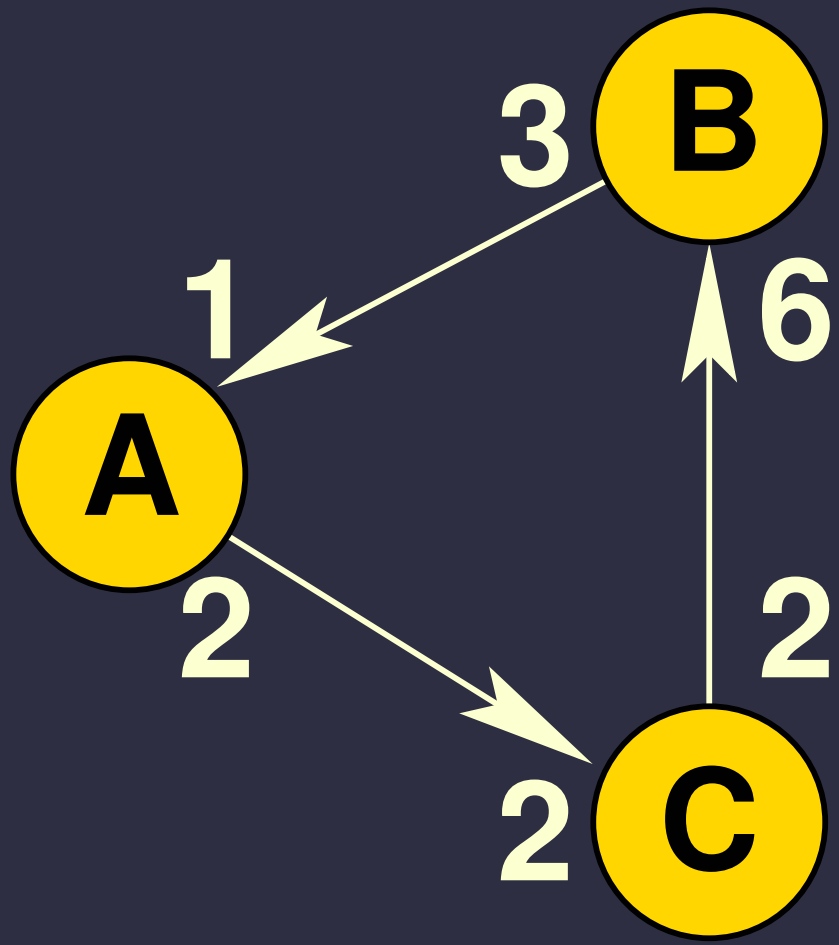
Synchronous dataflow graph (SDFG)



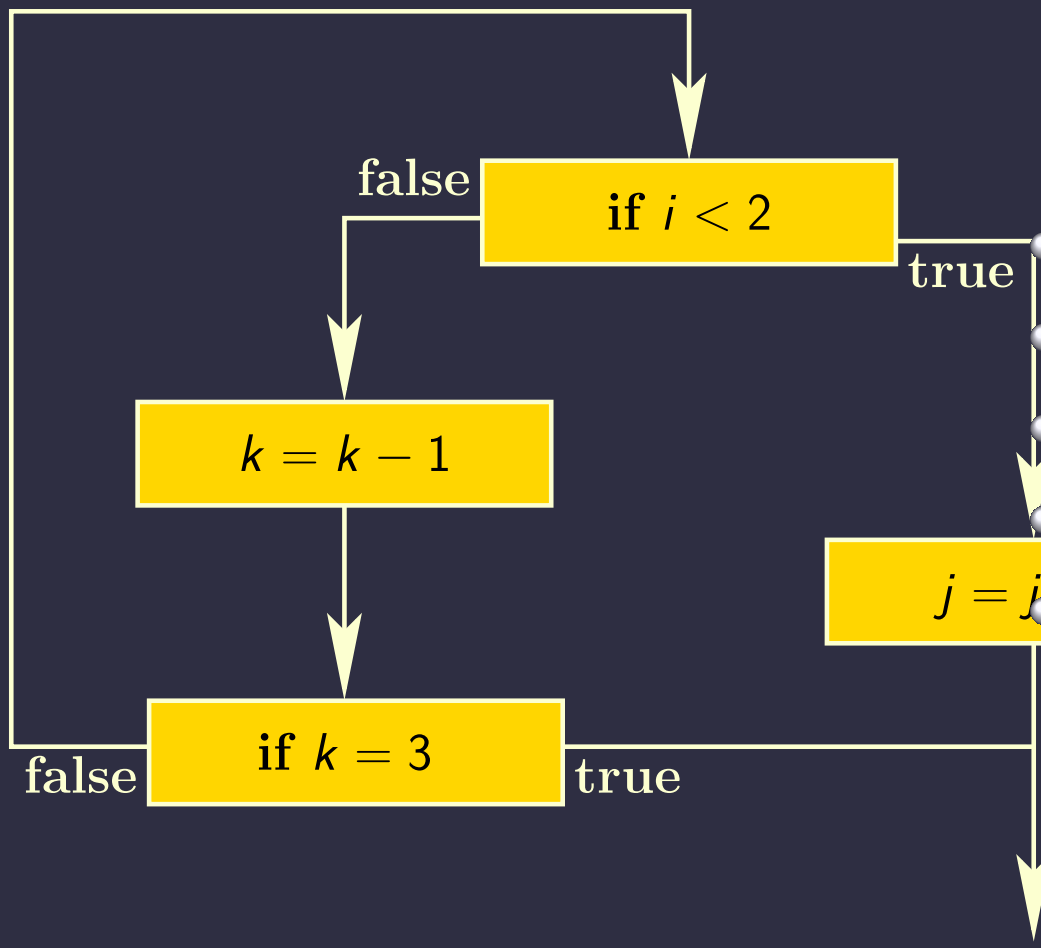
Synchronous dataflow graph (SDFG)



Synchronous dataflow graph (SDFG)

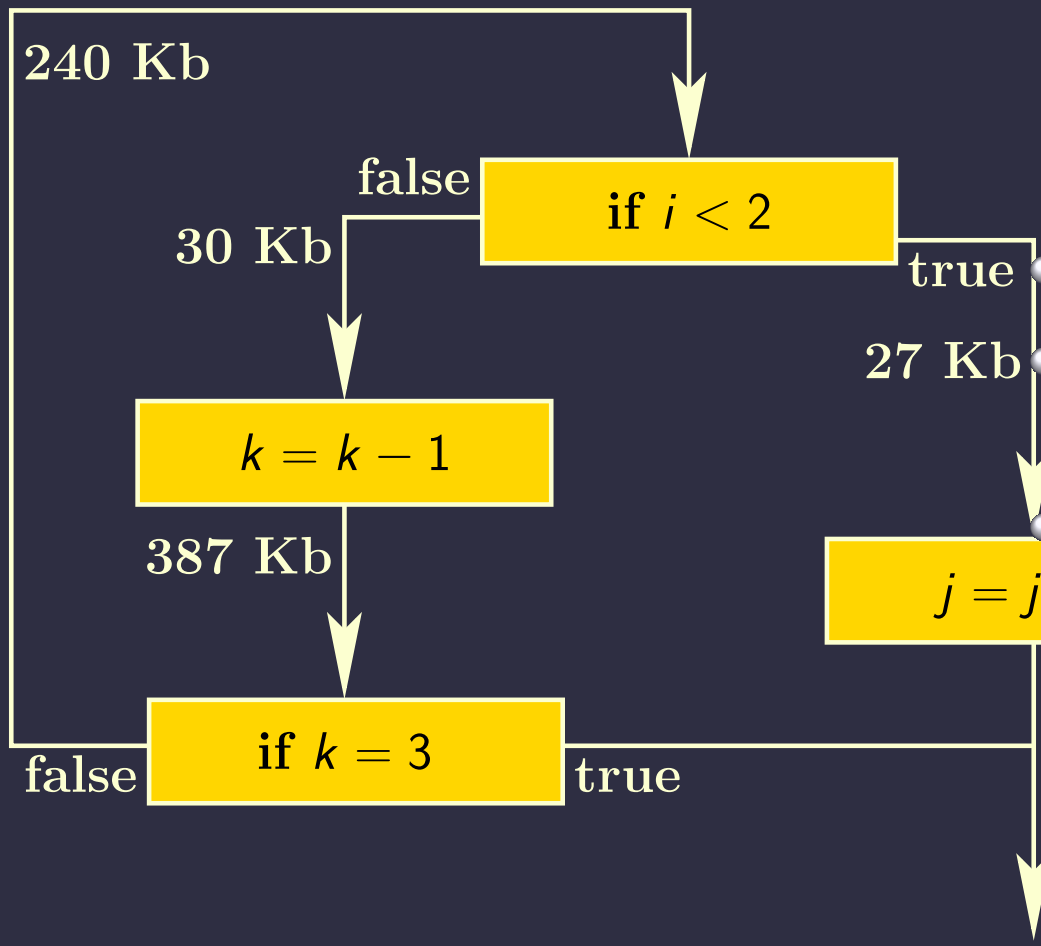


Control flow graph (CFG)



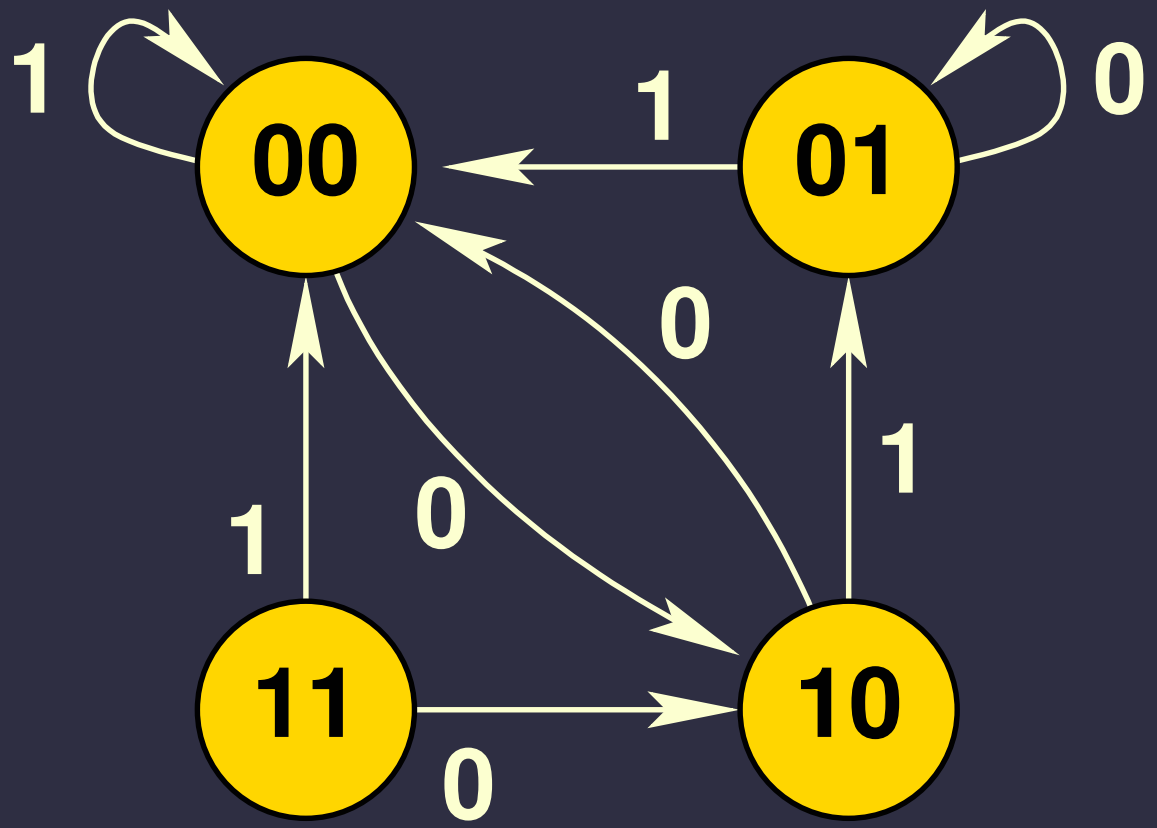
- Nodes are tasks
- Supports conditionals, loops
- No communication quantities
- SW background
- Often cyclic

Control dataflow graph (CDFG)



- Supports conditionals, loops
- Supports communication quantities
- Used by some high-level synthesis algorithms

Finite state machine (FSM)



Finite state machine (FSM)

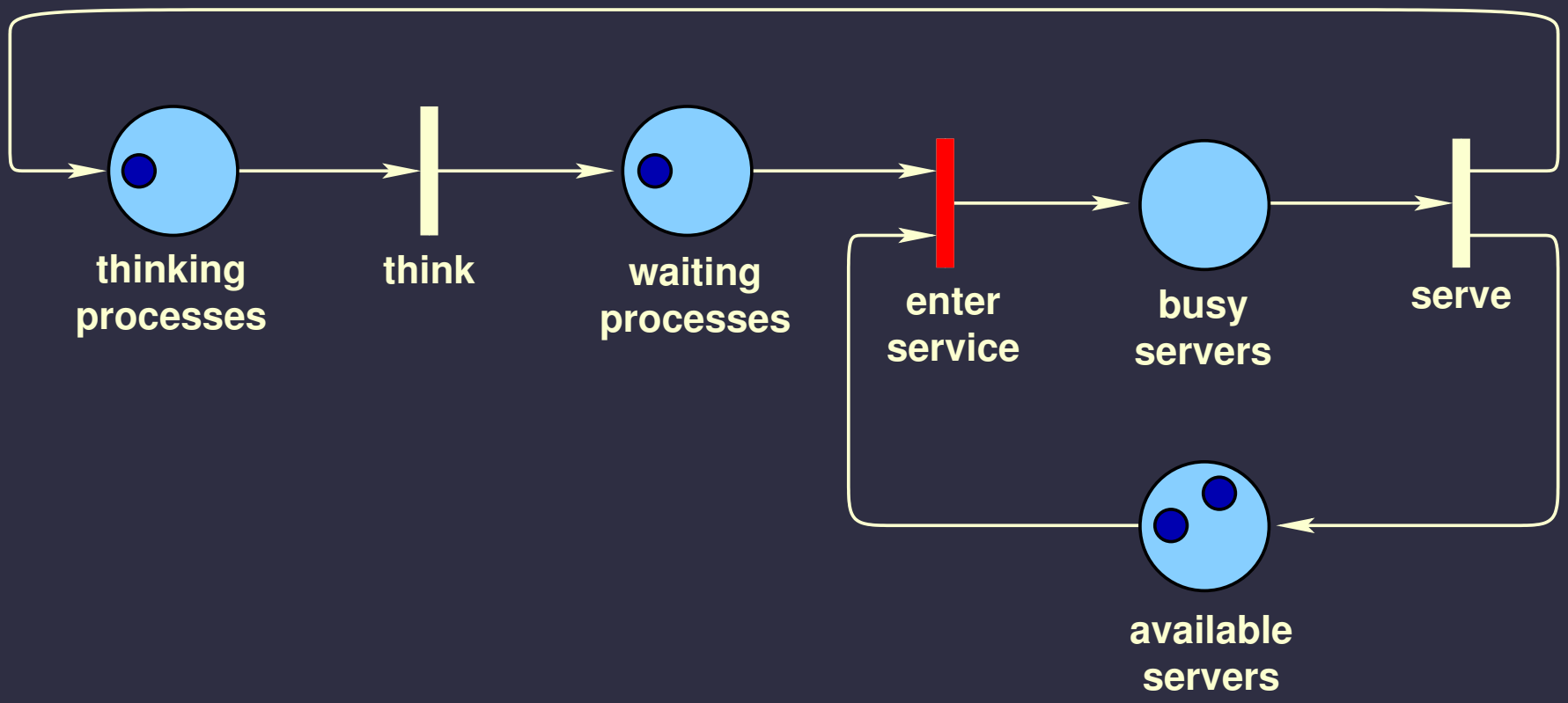
	input	
	0	1
00	10	00
01	01	00
10	00	01
11	10	00
current	next	

- Normally used at lower levels
- Difficult to represent independent behavior
  - State explosion
- No built-in representation for data flow
  - Extensions have been proposed
- Extensions represent SW, e.g., co-design finite state machines (CFSMs)

Petri net

- Graph composed of places, transitions, and arcs
- Tokens are produced and consumed
- Useful model for asynchronous and stochastic processes
- Places can have priorities
- Not well-suited for representing dataflow systems
- Timing analysis quite difficult
- Large flat graphs difficult to understand

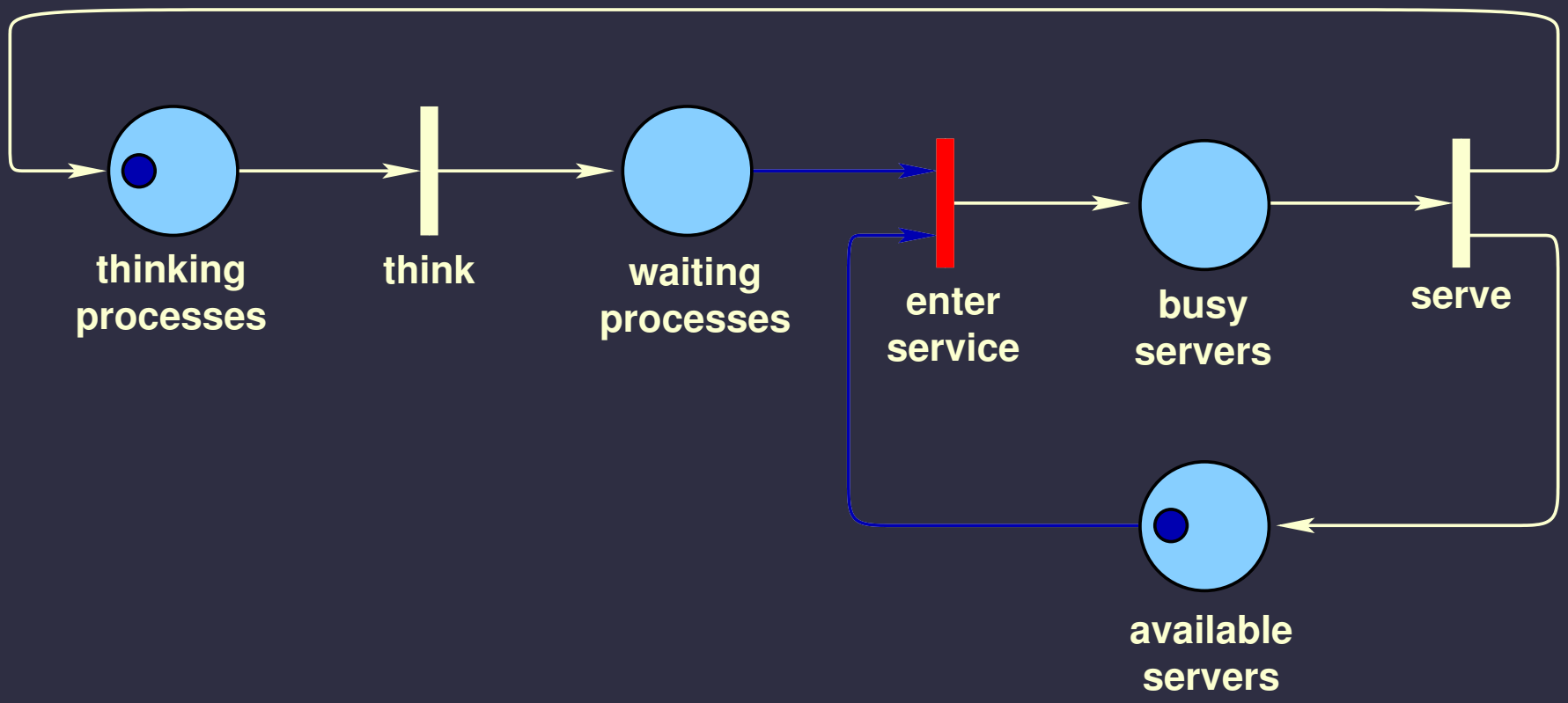
Petri net



M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann's token game demonstration.

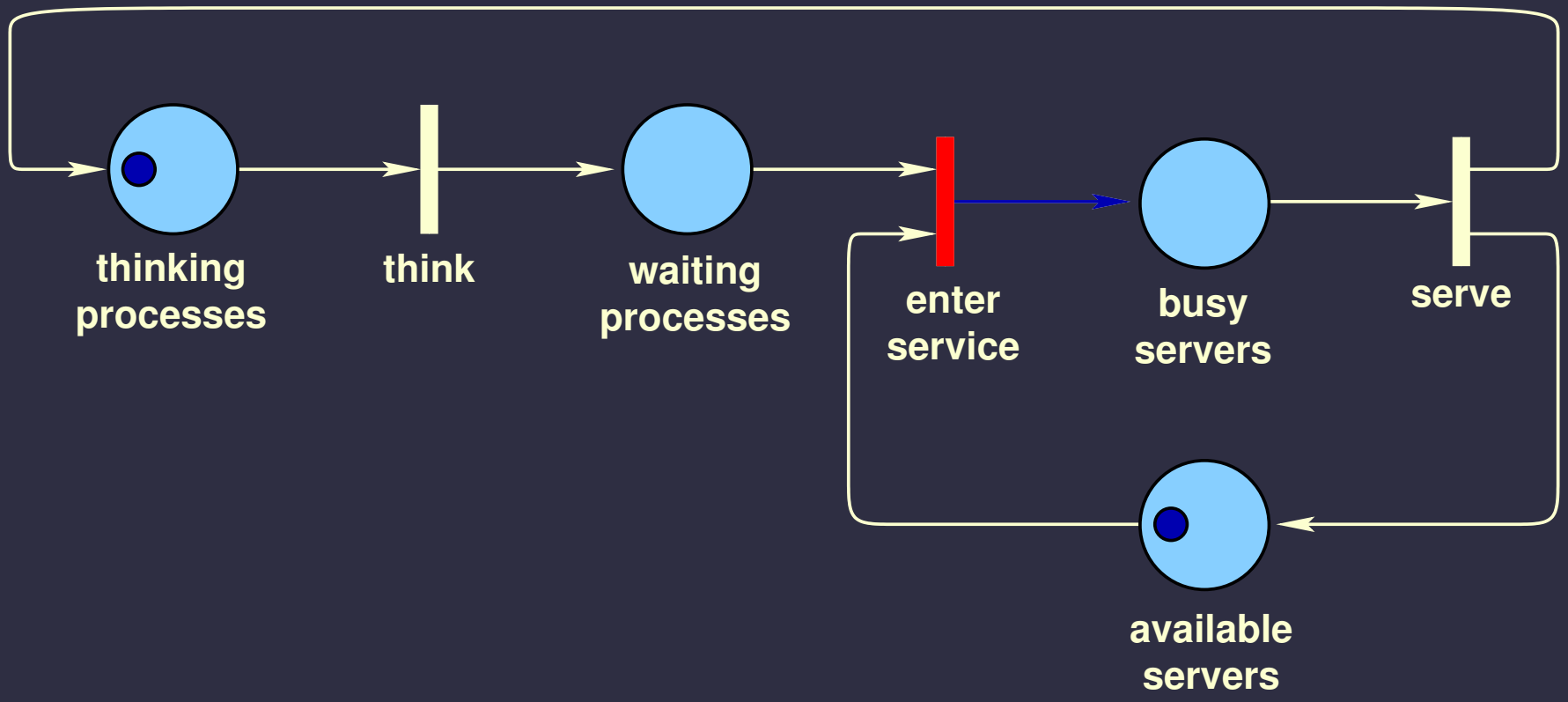
Petri net



M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann's token game demonstration.

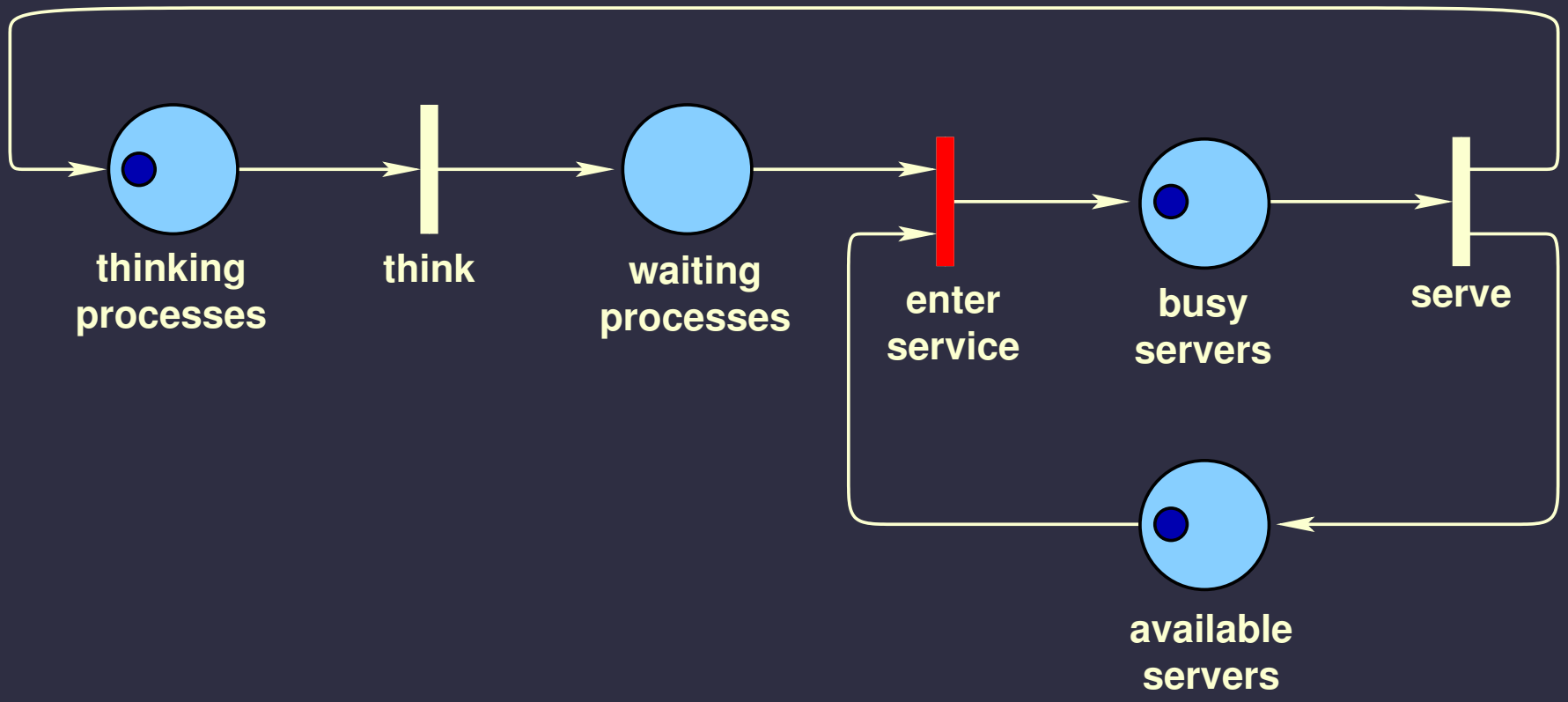
Petri net



M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann's token game demonstration.

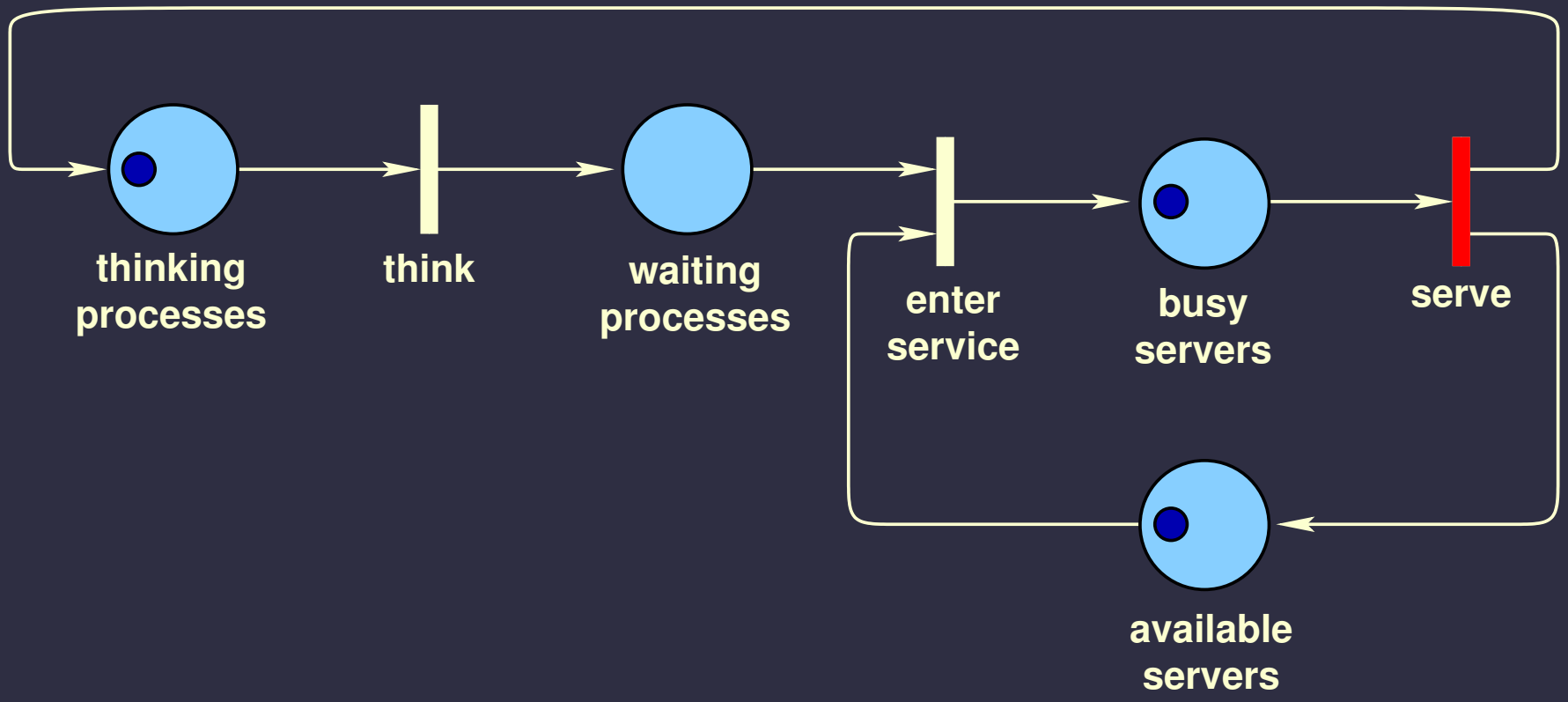
Petri net



M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann's token game demonstration.

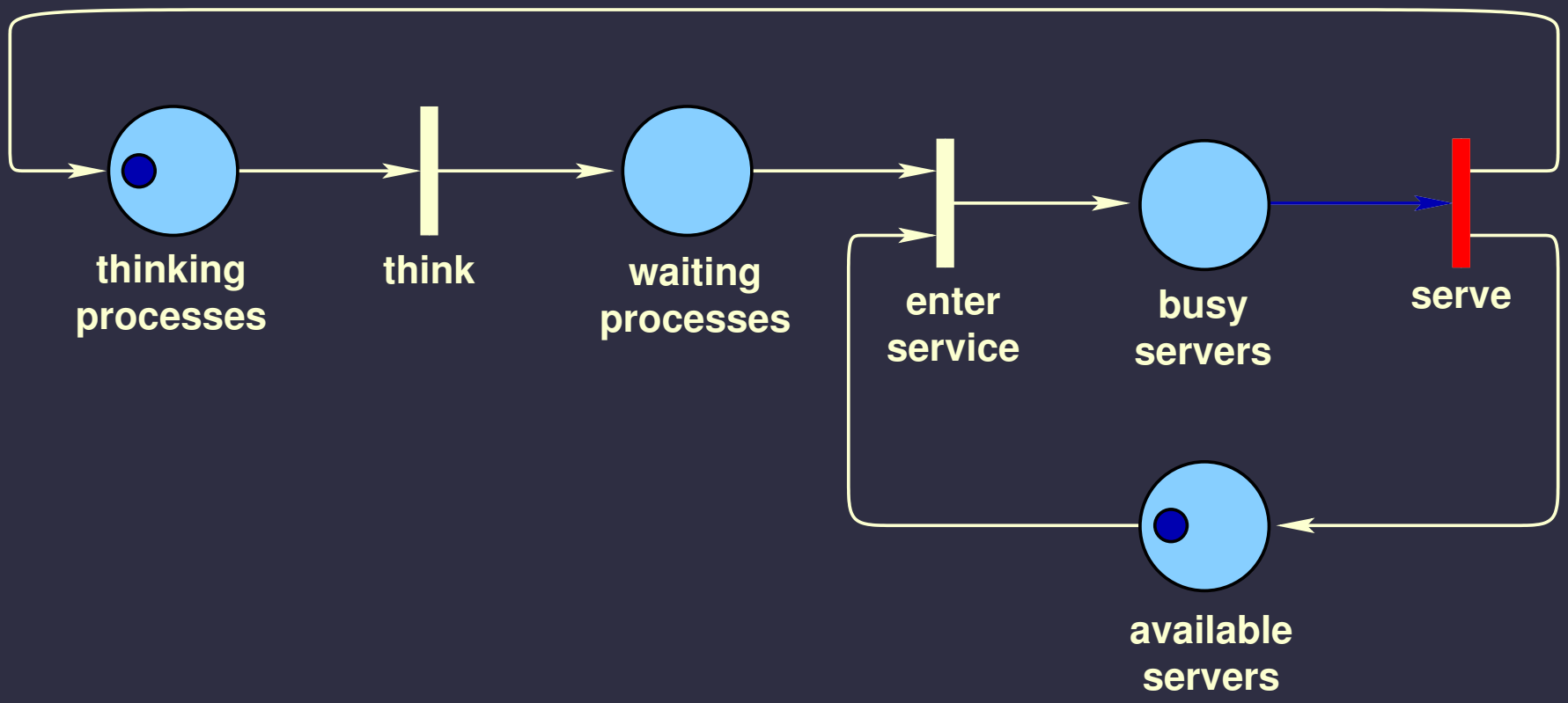
Petri net



M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann's token game demonstration.

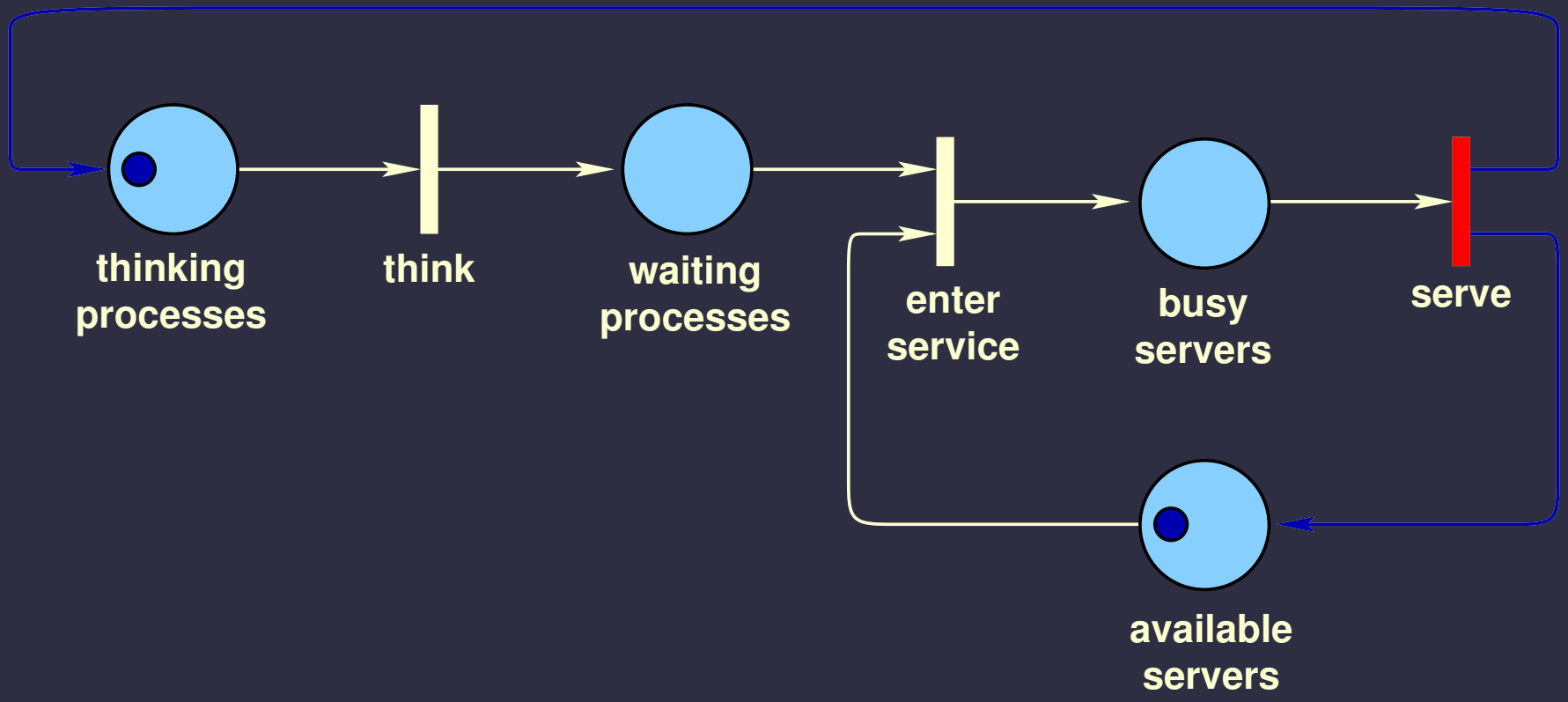
Petri net



M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann's token game demonstration.

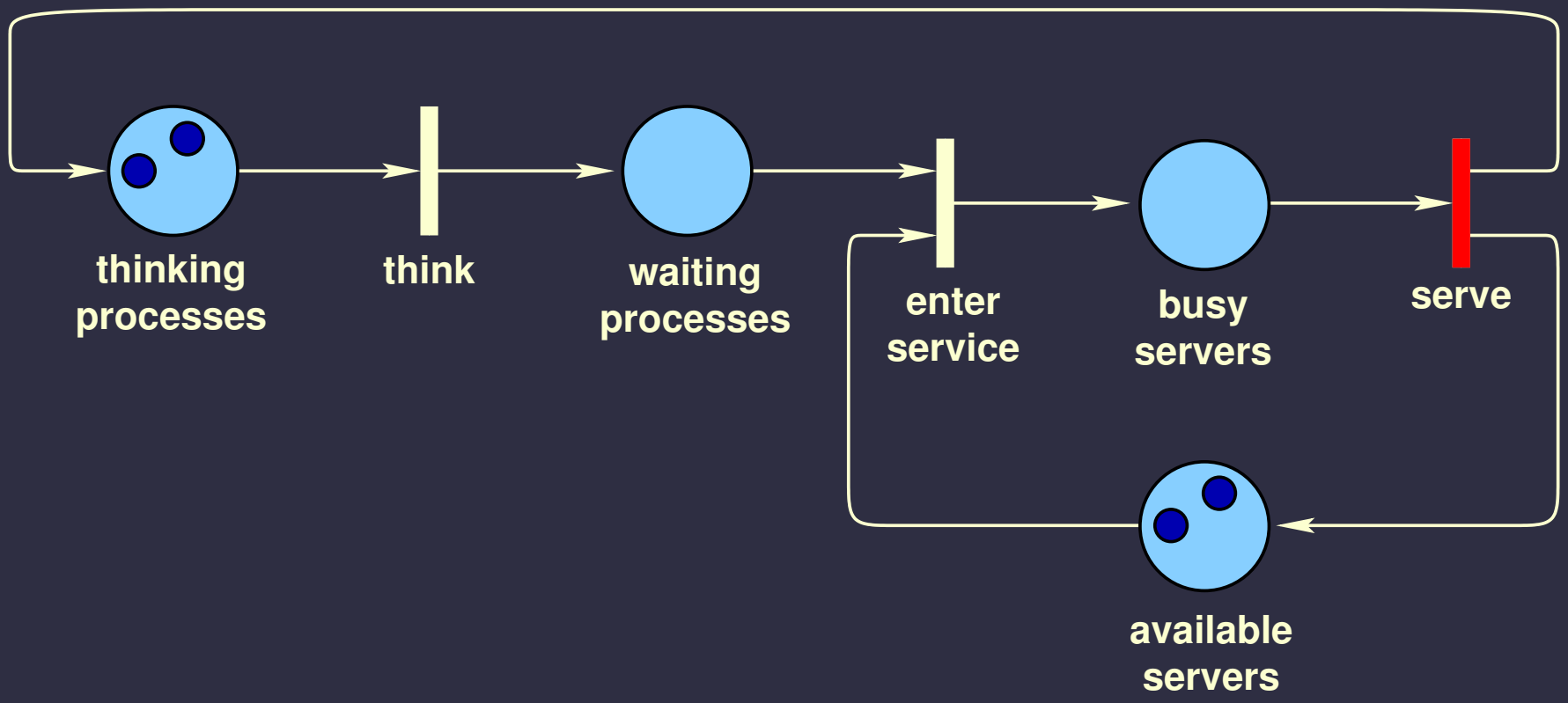
Petri net



M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann's token game demonstration.

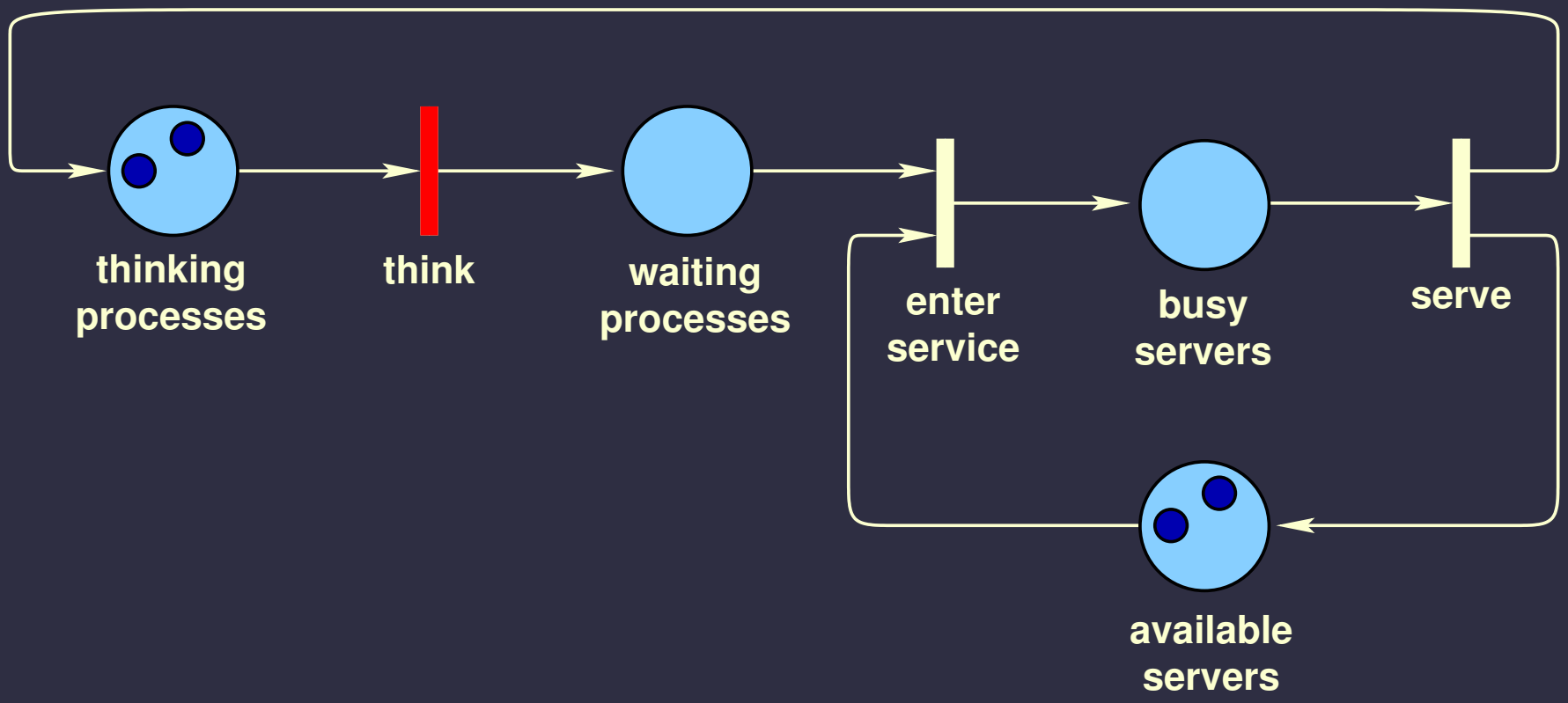
Petri net



M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann's token game demonstration.

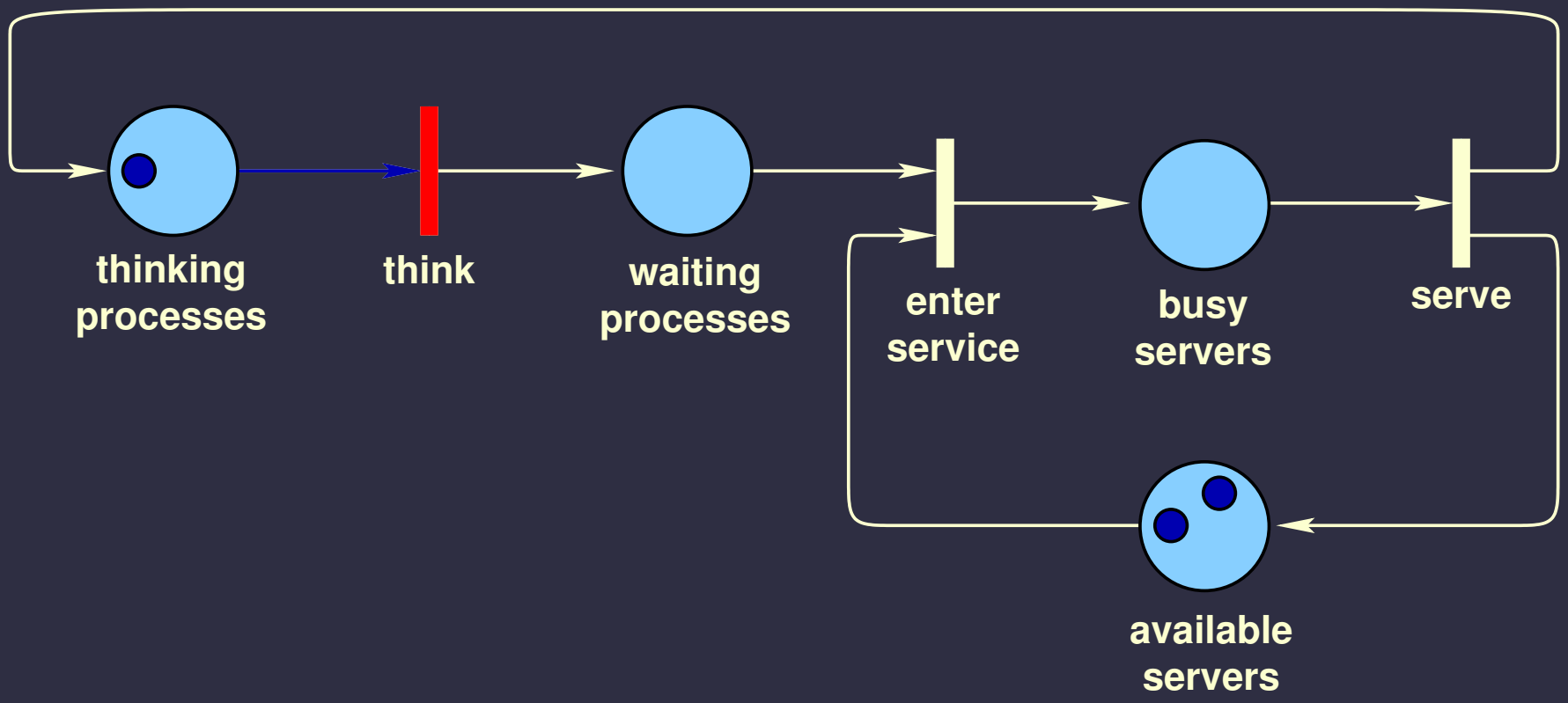
Petri net



M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann's token game demonstration.

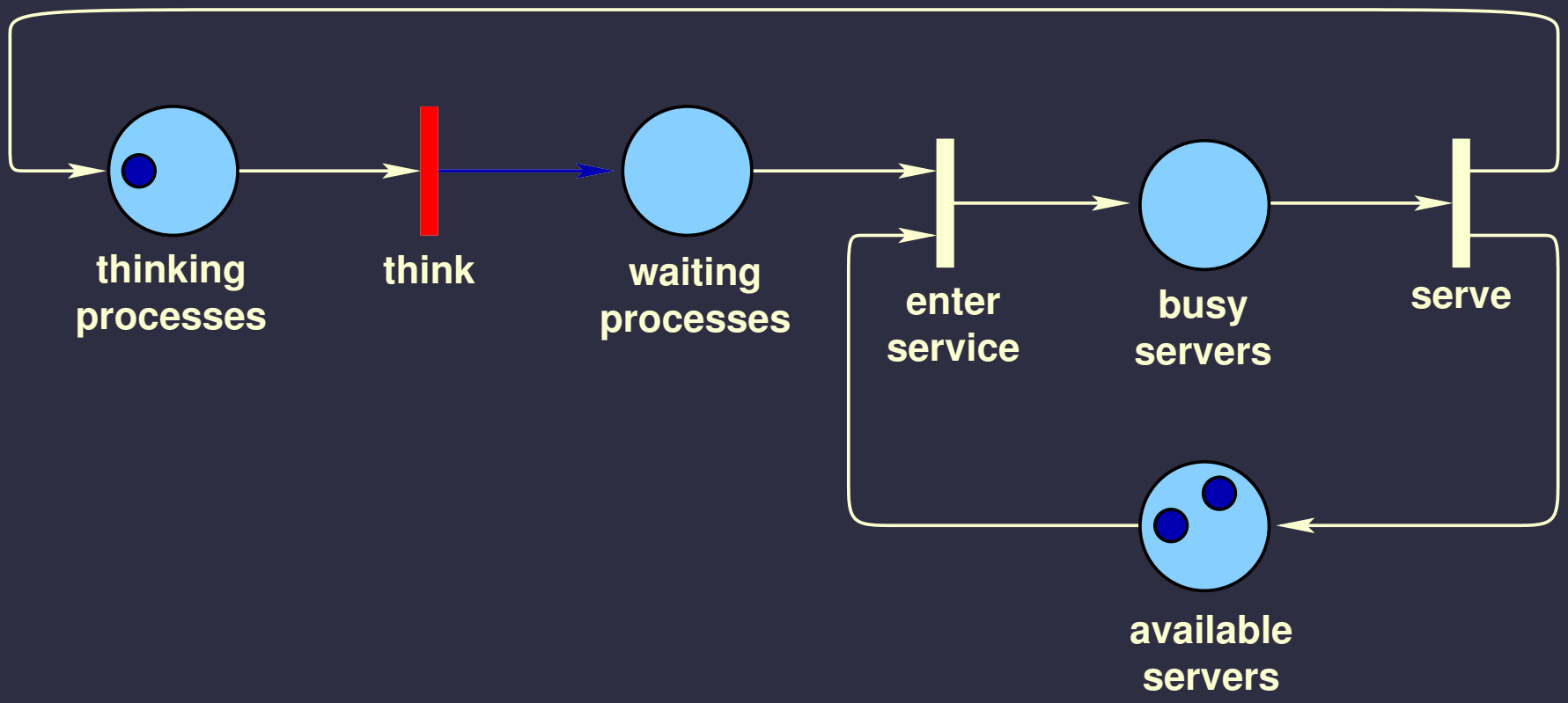
Petri net



M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann's token game demonstration.

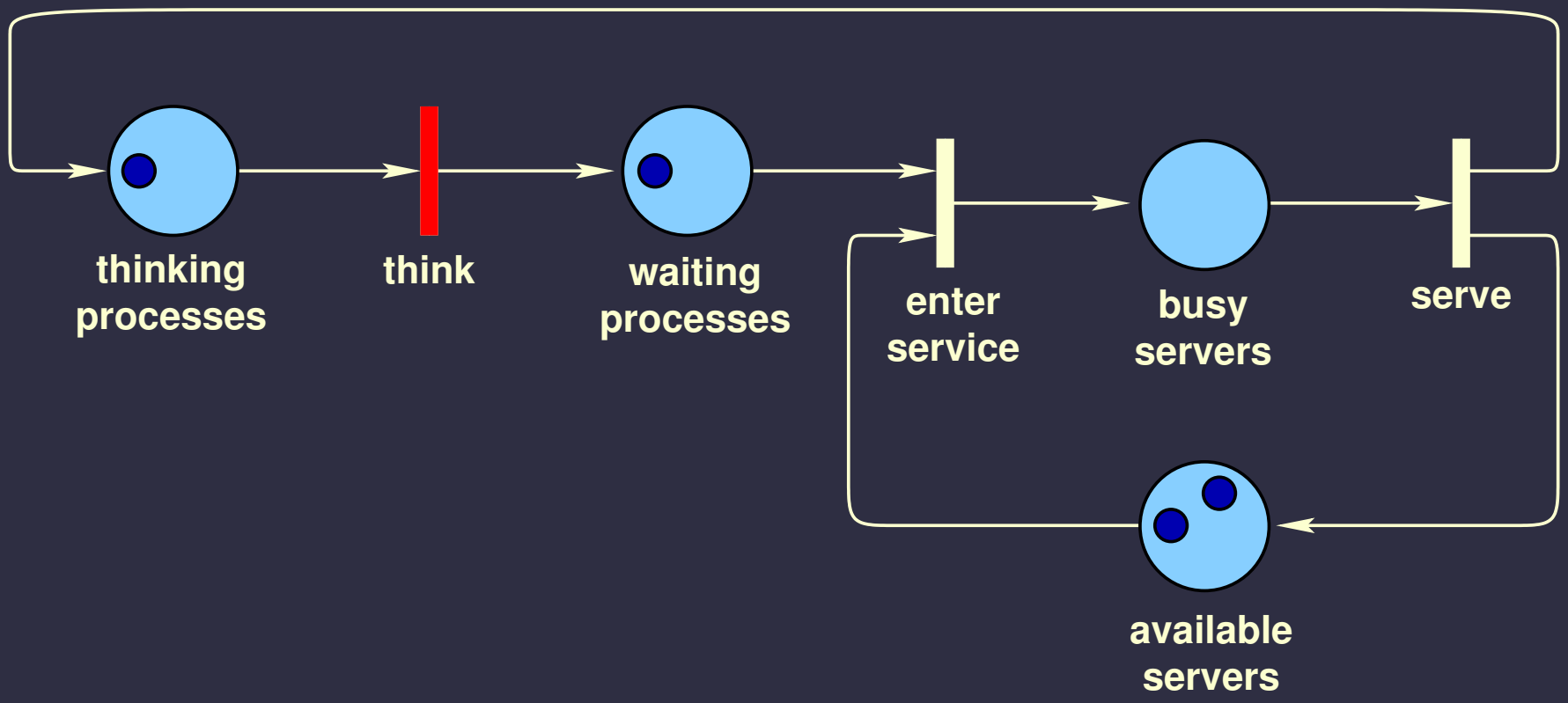
Petri net



M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann's token game demonstration.

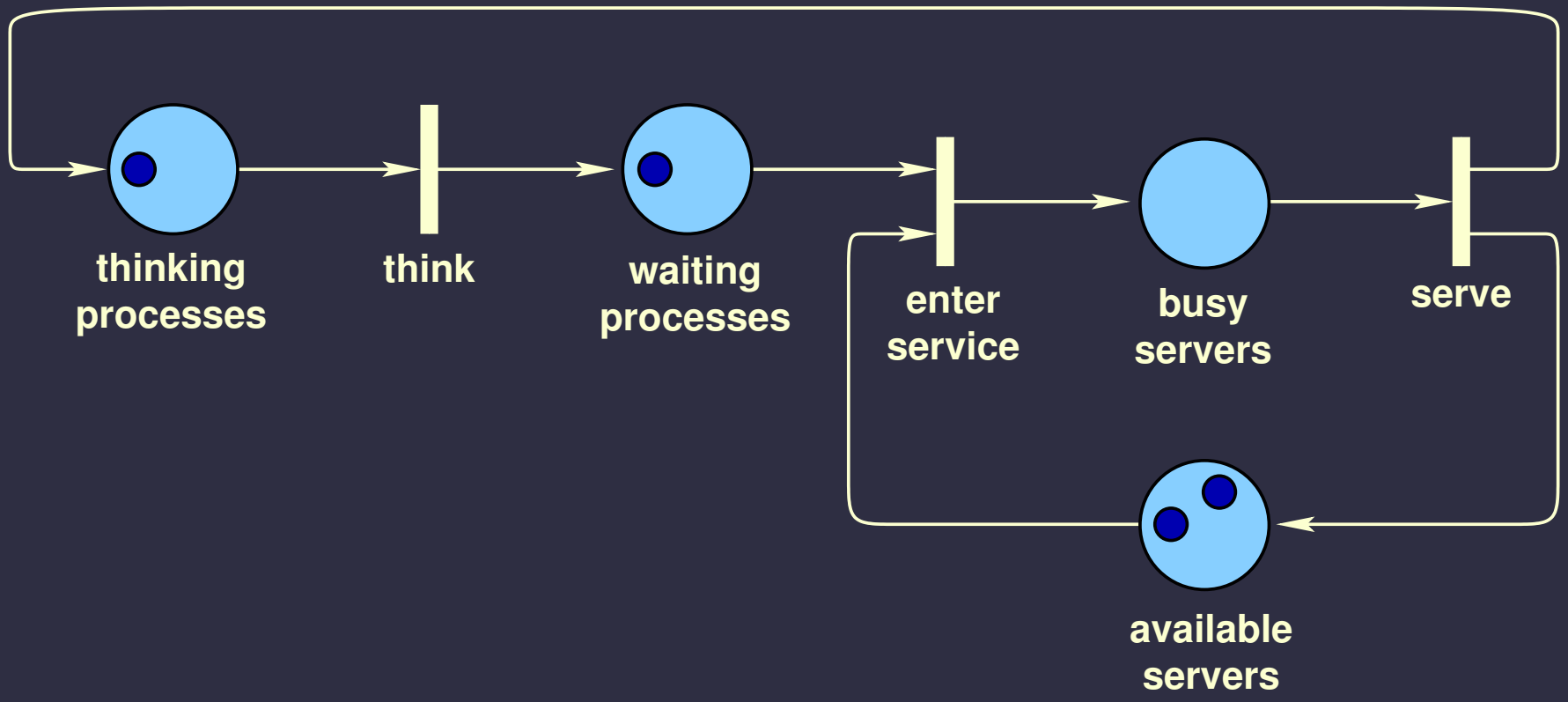
Petri net



M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann's token game demonstration.

Petri net



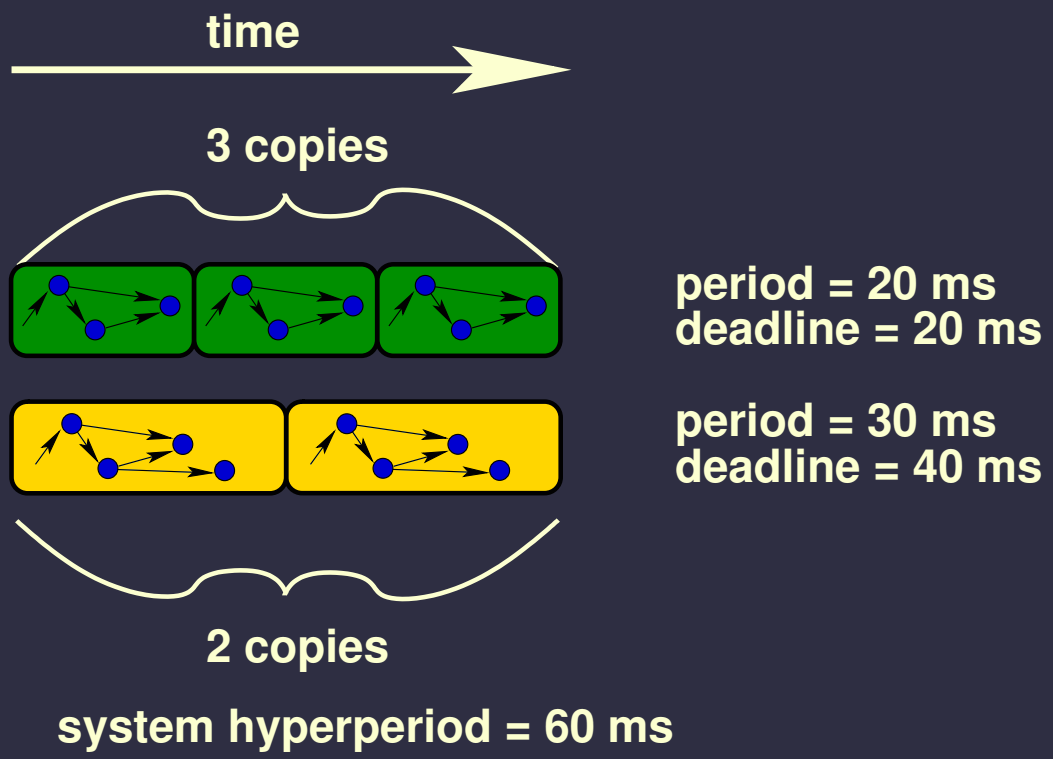
M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann's token game demonstration.

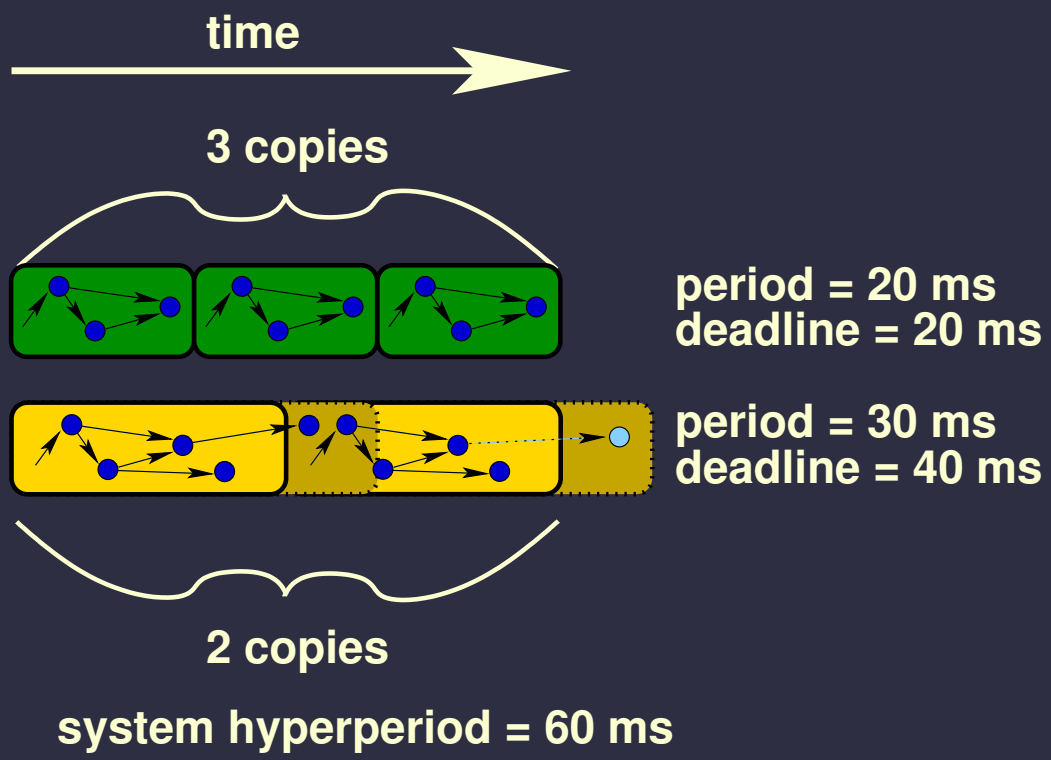
Periodic graphs

- Some system specifications contain periodic graphs
- Can guarantee scheduling validity by scheduling to the least common multiple of periods
- Can also meet aperiodic specifications, however, resources will sometimes be idle

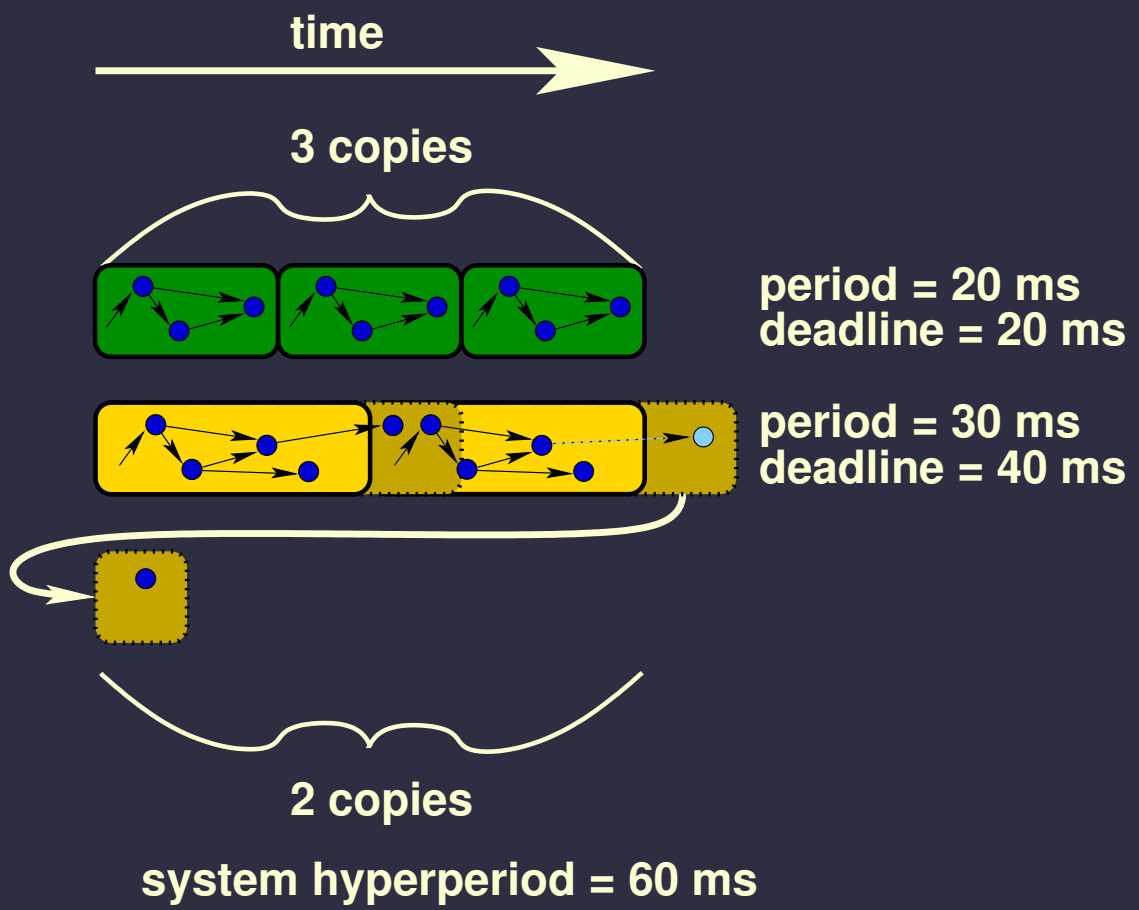
Periodic graphs



Periodic graphs



Periodic graphs



Aperiodic graphs

- No precise periods imposed on task execution
- Useful for representing reactive systems
- Difficult to guarantee hard deadlines in such systems
  - Possible if minimum inter-arrival time known

Periodic vs. aperiodic

Periodic applications

- Power electronics
- Transportation applications
  - Engine controllers
  - Brake controllers
- Many multimedia applications
  - Video frame rate
  - Audio sample rate
- Many digital signal processing (DSP) applications

However, devices which react to unpredictable external stimuli have aperiodic behavior

Many applications contain periodic and aperiodic components

Aperiodic to periodic

Can design periodic specifications that meet requirements posed by aperiodic specifications

- Some resources will be wasted

Example:

- At most one aperiodic task can arrive every 50 ms
- It must complete execution within 100 ms of its arrival time

Aperiodic to periodic

- Can easily build a periodic representation with a deadline and period of 5 ms
  - Problem, requires a 50 ms execution time when 100 ms should be sufficient
- Can use overlapping graphs to allow an increase in execution time
  - Parallelism required

The main problem with representing aperiodic problems with periodic representations is that the tradeoff between deadline and period must be made at the time of synthesis

Real-time vs. best effort

- Why make decisions about system implementation statically?
  - Allows easy timing analysis, hard real-time guarantees
- If a system doesn't have hard real-time deadlines, resources can be more efficiently used by making late, dynamic decisions
- Can combine real-time and best-effort portions within the same specification
  - Reserve time slots
  - Take advantage of slack when tasks complete sooner than their worst-case finish times

Discrete vs. continuous timing

System-level: continuous

- Operations are not small integer multiples of the clock cycle

High-level: discrete

- Operations are small integer multiples of the clock cycle

Implications:

- System-level scheduling is more complicated. . .
- . . . however, high-level also very difficult.

Section outline

Processing resource description

- Often table-based
- Price, area
- For each task
  - Execution time
  - Power consumption
  - Preemption cost
  - etc.
- etc.

Similar characterization for communication resources

Wise to use process-based

Communication resource description

- Can use bus-bridge based models for distributed systems
- Wireless models
- etc.
- However, in the future, it will become increasingly important to base SOC communication model on process parameters

Example from research

Data-flow graphs

- Multirate
- Hard real-time
- Some aperiodic hard real-time tasks
- May have best-effort tasks

System-level representations summary

- No single representation has been decided upon
- Software-based representations becoming more popular
- System-level representations will become more important
- This is still an active area of research

Notes on clustering and partitioning

- Interdependence with architecture
- Heterogeneity's impact on partitioning
- Applications to grid computing
- Dynamic partitioning

Interesting future direction

Open problem

- Can specification be so simple for some embedded application domains that application experts who are not computer engineers easily do it?
- What HCI, compiler, and synthesis support is required?

Begin study of topics of interest

- Due in class on 8 September.
- Use electronic resources, research papers, and questions posted to the mailing list to answer the following questions for each of the three topics of interest, using three or fewer sentences for each.
  1. How useful will this be to designers in the next ten years?
  2. Is this topic of special interest to embedded system designers?
  3. Identify a potential research project that is related to this topic and can be completed within the time-frame of this course.
- We will discuss your answers in the next class.

Outline

Decide mini-projects

By 13 September, define a mini-project.

- Goal.
- Formal problem statement.
- Most closely related work (3–5).
- What will be completed within mini-project.
- Can form teams now or on 13 September.

I will send comments on today's assignment via email.

We can discuss your project ideas before Monday.

Areas of interest I

5: Manual and automated design of distributed sensing applications.

5: Power and energy

- 5: Low-power design and power management.
- 1: Energy supply in embedded systems, including energy harvesting.

4: Real-time and reliability

- 2: Real-time and embedded operating systems
- 2: Real-time systems and scheduling
- 2: Embedded system reliability

Areas of interest II

- 3 Wireless communication among heterogeneous distributed embedded systems.
- 2: Autonomous control and coordination of low-power robots.
- 2: Manual or automated design with heterogeneous components.
- Home automation and security applications.
- Human-driven computer design.
- Making embedded systems extremely compact.
- Medical applications.
- Portable communication applications.

Areas of interest III

- Design with off-the-shelf components.
- Distributed embedded systems for supercomputing.
- Design of coprocessor systems for improved performance and efficiency.
- Using embedded systems to undermine the security of other systems.
- Simplifying the embedded system design process.
- Automotive applications.
- Multimedia applications.

Upcoming topics

- Example project on specification languages.
- Overview of optimization techniques.
- Distributed system synthesis.