# The Control Point Policy

Stanley B. Gershwin

gershwin@mit.edu
http://web.mit.edu/manuf-sys

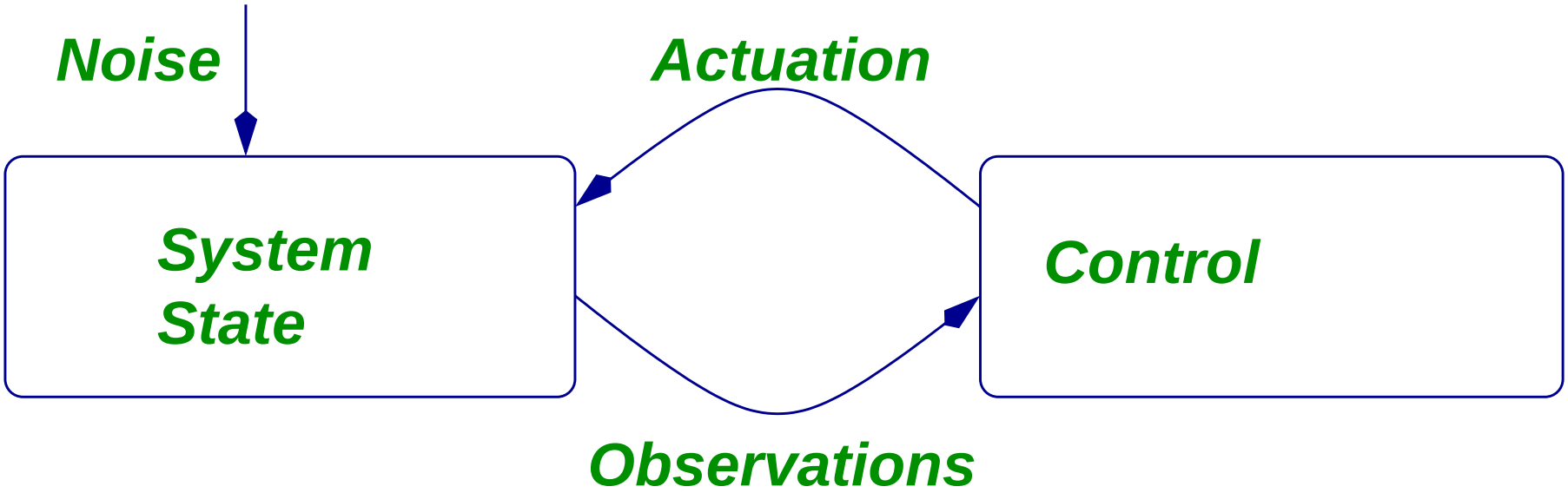
Massachusetts Institute of Technology

Spring, 2012

# Real-Time Scheduling

- Scheduling is the selection of times for future controllable events.
- Because of recurring random events, scheduling is an on-going process, and not a one-time calculation.
- The purpose of real-time scheduling in a factory is to make decisions in response to random events about the movement of material and scheduling of machines in a complex factory.

# Real-Time Scheduling
## Control Paradigm




This is the general paradigm for control theory and engineering.

# Real-Time Scheduling

In a factory,

- *State:* distribution of inventory, repair/failure states of machines, etc.
- *Control:* move a part to a machine and start operation; begin preventive maintenance, etc.
- *Noise:* machine failures, change in demand, etc.

# Real-Time Scheduling
## Requirements

Scheduling systems or methods should ...

- deliver good factory performance.
- compute decisions quickly, in response to changing conditions.

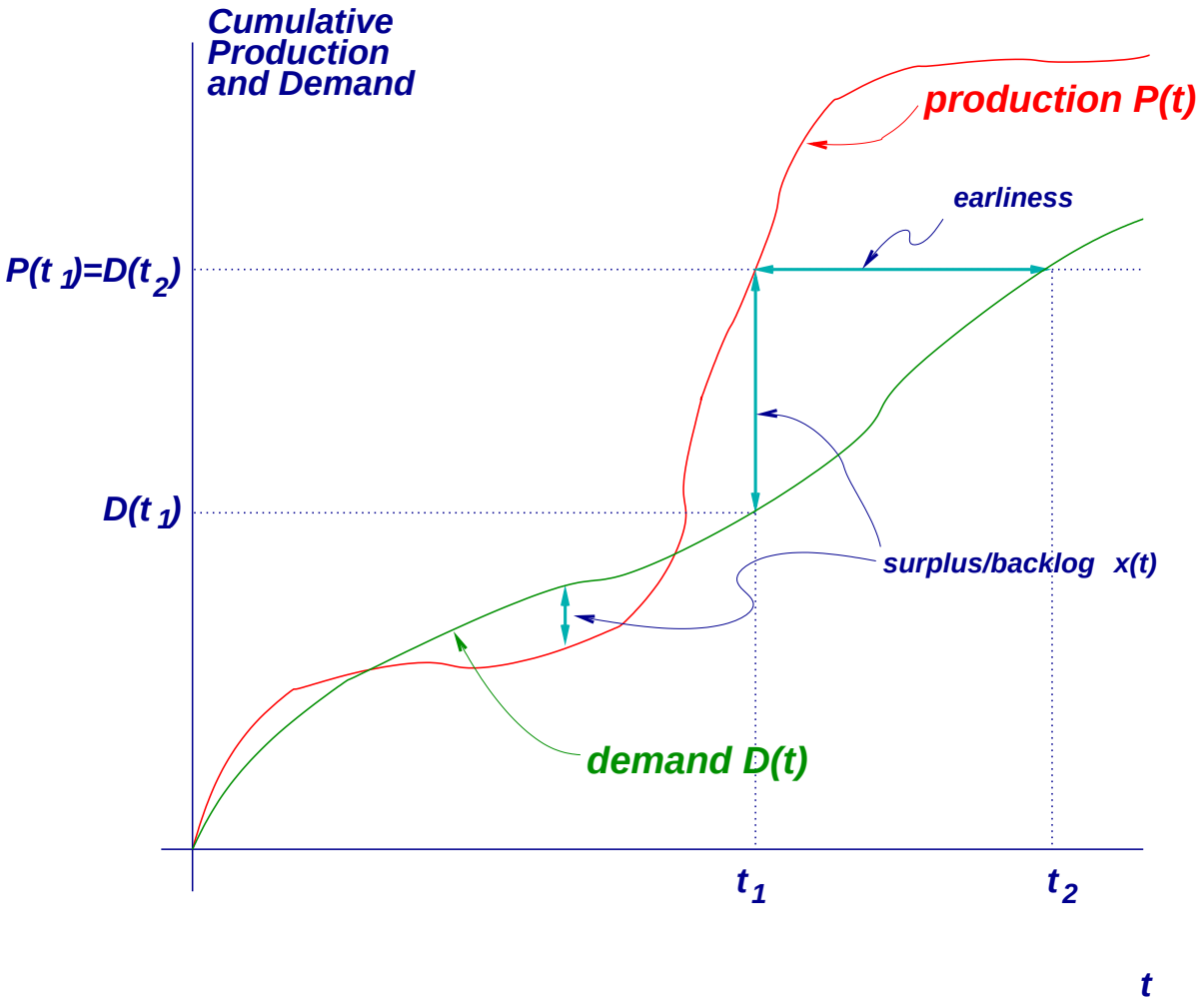
# Real-Time Scheduling
## Performance Goals

- Factory performance is measured by:
  - throughput;
  - turn-around time;
  - inventory;
  - backlog;
  - due date reliability.
  - ... and other quantities.
- To maximize the probability that customers are satisfied.
- To maximize predictability (ie, minimize performance variability).
- To provide better service for more important products.

# Real-Time Scheduling
## Design Goals

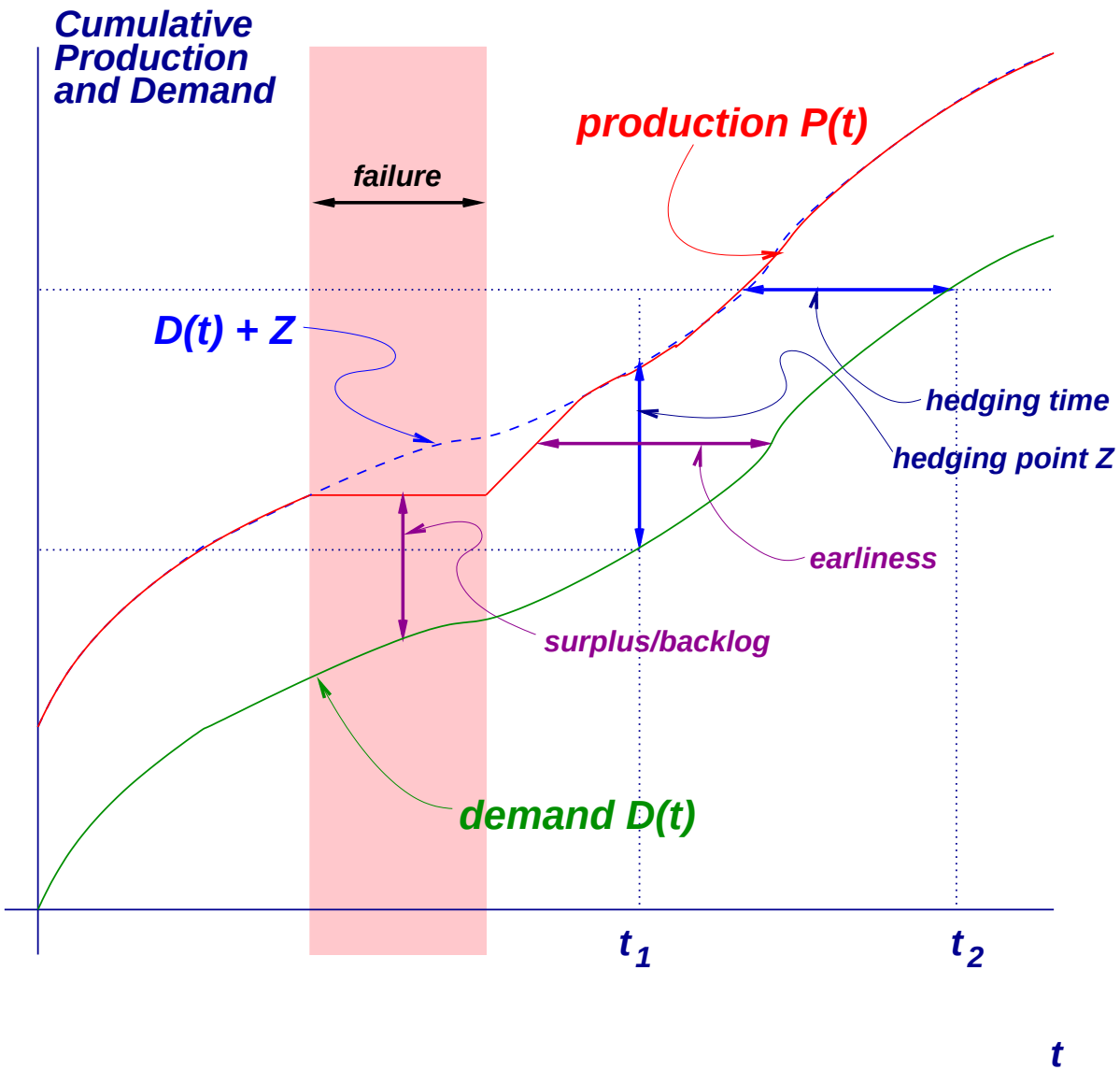
- To be easy to implement.
- To be easy to use.
- To be easy to modify.
- To respond to random events immediately, including
  - Machine failures,
  - Machine repairs,
  - Scrapping of material,
  - Demand spikes,
  - Material handling traffic congestion,
  - etc.

# Real-Time Scheduling
## Objective of Scheduling




*The objective is to keep cumulative production close to cumulative demand.*

# Real-Time Scheduling
## Hedging point policy

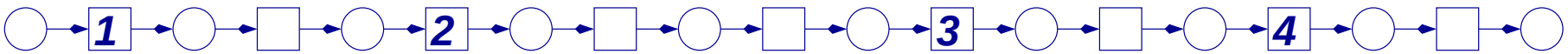



*Hedging point policy:*
Try to keep production above demand by $Z$; try to keep $P(t) = D(t) + Z$.

When machine fails, production stops. When machine is repaired, produce at maximum rate until $P(t) = D(t) + Z$ is satisfied.

# The Control Point Policy

- The *control point policy* is the hedging point policy applied at one or more control points along a line.




- It can be implemented in three ways:
  - as a surplus-based policy
  - as a token-based policy
  - as a time-based policy

# The Control Point Policy

- Our analytical methods will make it possible to:
  - predict performance, and
  - choose hedging points (or hedging times) and buffer sizes to optimize performance.
- This can be done by doing numerical calculations, *not* by running simulations.
  - Designs are therefore *better* and obtained *much faster* than by using simulation.

# Control Point Policy Philosophy
## The CPP takes into account ...

- The relative importance of products.
- The due date for each item.
- The remaining work time and expected remaining production lead time for each item.

# Control Point Policy Philosophy
## The CPP obtains good performance because ...

- It only advances items when they will not have to wait long for operations.
  - This keeps inventory and lead time small, and it prevents downstream congestion.
- It resequences parts at each control point as needed.
  - More important parts, and parts in greatest danger of missing due dates are moved up in the queue.
- Real-time calculations are trivial
  - There are no heavy computational burdens or long computational delays.
- It responds immediately to random events such as breakdowns.

# CPP Assumptions

We are managing a factory in which

- There is a limited set of paths that material follows (possibly only one).
- There is a set of *importance classes* of material (possibly only one).
- Products have due dates.
- Disruptions occur at random times.

# CPP Features

- The CPP decisions can be based on *time*, *tokens*, or *surplus* .
- There are a set of *control points* that limit the flow of material into and within the system.
- At each control point,
  - the policy resequences material to respond to machine failures or other disruptions. The parts are resequenced in order of importance and danger of missing due dates.
  - material is prevented from moving downstream by limiting (1) how early the material is and (2) how much inventory is downstream.
- Propagation of disruptions is reduced by storage space and resequencing.

# CPP Features

There is a *preparation* phase and a *real-time* phase.

- *Preparation phase* :
  - Select control points.
  - Rank order the parts
  - Determine *hedging parameters* (described below).
  - determine buffer sizes.
- *Real-time phase* :
  - Described below.

# CPP Definitions — *Time-Based Version*

- A *control point* is a machine where we apply the policy fully.
- A *hedging time* is the time that we allow a part between a control point and the end of the process (ie, the shipping dock).
  - *a conservative estimate of* the lead time that takes into account the possibilities of delay due to machine failure and queuing; and takes into account the risk of late deliveries and the cost of inventory. It is:
  - *NOT* the expected lead time — greater than the expected lead time.
  - greater than the minimal time for remaining process.

For example, the hedging time could be the mean + 2 * standard deviation of the lead time.

# CPP Definitions — *Time-Based Version*

- A machine is *available* if it is not performing an operation, not undergoing repair or maintenance, and not otherwise occupied.
- A part is *available* at a control point if it is present and has had the previous operation, and if it is not blocked downstream.
- A part is *ready* at a control point if it is available and

  $$\boxed{\text{the current time} + \text{the hedging time} > \text{the due date.}}$$

  Example:
  - At Machine X (a control point), the hedging time for Type Y parts is 20 days and 3 hours. If the current time is 11:00 AM on July 1, a Type Y part is ready if its due date is earlier than 2:00 PM on July 21.

# CPP Real-Time Phase

In real time:

- At each control point, whenever the machine becomes available,
  - Find the highest ranking part type that is ready. Load it and work on it immediately.
  - If no available parts are ready, wait. Look at the system again when either another part arrives from upstream or one of the available parts becomes ready, whichever comes first. Then go to the previous step.
- At each other machine, do something simple like first-in-first-out. Do not allow the resource to be idle when there are parts that can be worked on.

# CPP Example

- In the remaining slides, we show an example of how parts are selected to be processed according to the time-based version of the CPP.
- We show the evolution of the state of a simple factory.
- Operations times are all 1.
- Failures are not modeled.
- There are three part types. Each has its own importance class.
- Each part has a due date.
- There are three control points.
- Each control point has three hedging times, one for each part type.

# CPP Example
## Notation

- Importance class 1 is the most important. If $i < j$, Class $i$ is more important than Class $j$.
- Part $i.n$ is the $n$th part of importance class $i$.
- $D(i.n)$ is the due date of Part $i.n$.
- $H(k, i)$ is the hedging time for class $i$ parts at control point $k$.

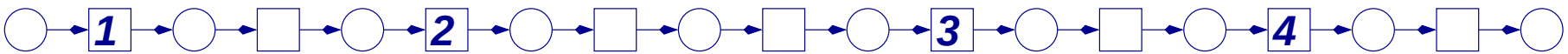
# CPP Example
## Data

Due dates: for the first three parts of the three part types,

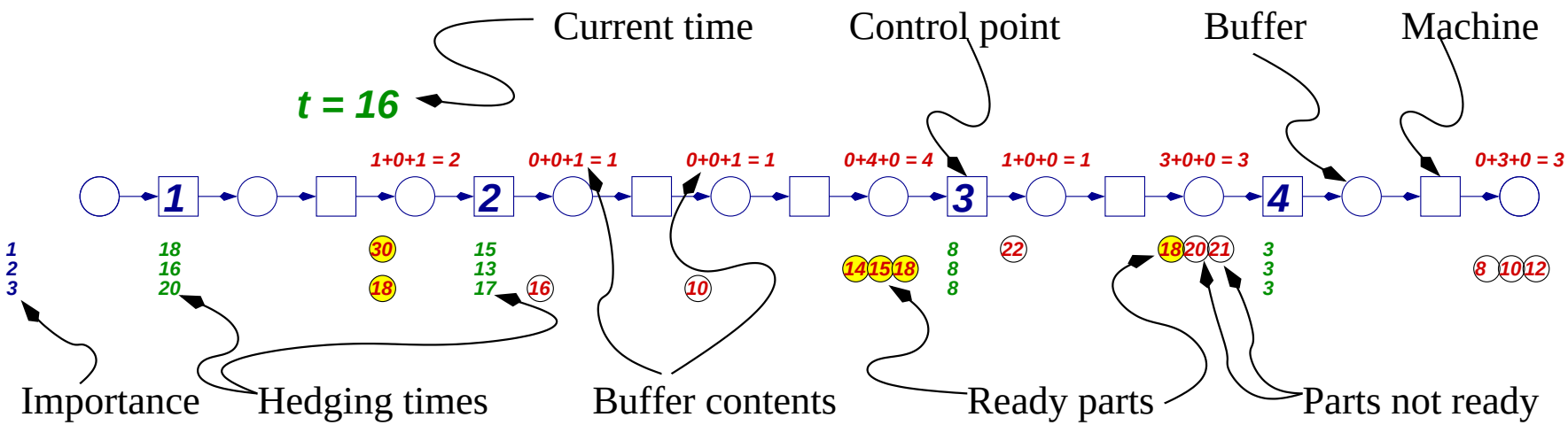
| $i.n$ | 1.1 | 1.2 | 1.3 | 2.1 | 2.2 | 2.3 | 3.1 | 3.2 | 3.3 |
|---|---|---|---|---|---|---|---|---|---|
| $D(i.n)$ | 18 | 20 | 30 | 8 | 12 | 15 | 10 | 16 | 18 |

Hedging times $H(k, i)$

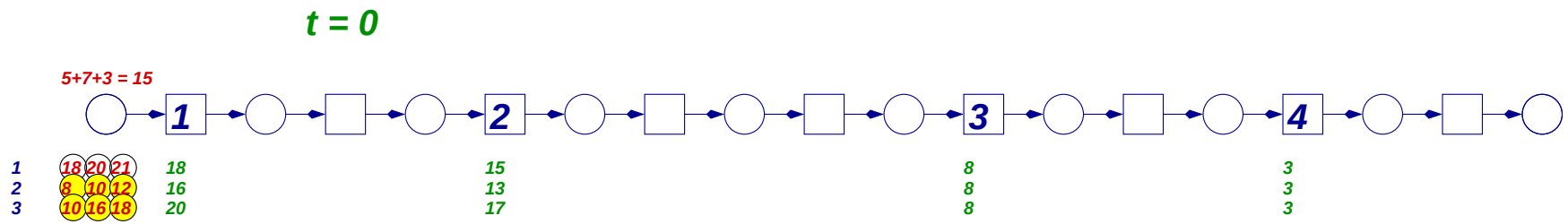
| $i \backslash k$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 18 | 15 | 8 | 3 |
| 2 | 16 | 13 | 8 | 3 |
| 3 | 20 | 17 | 8 | 3 |

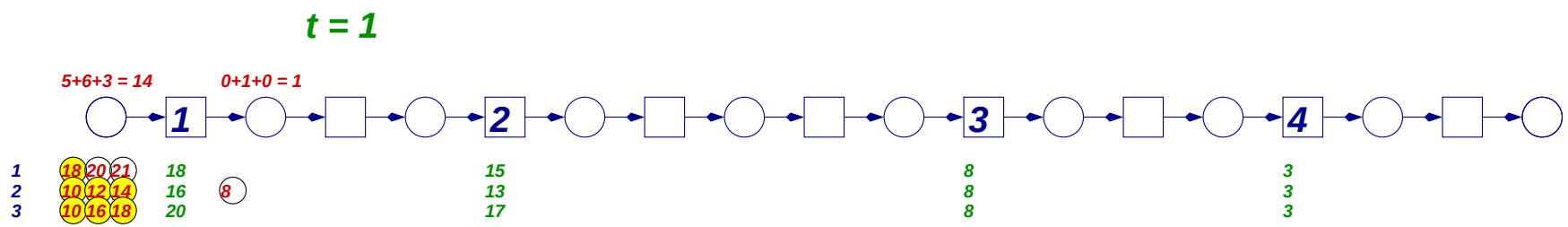


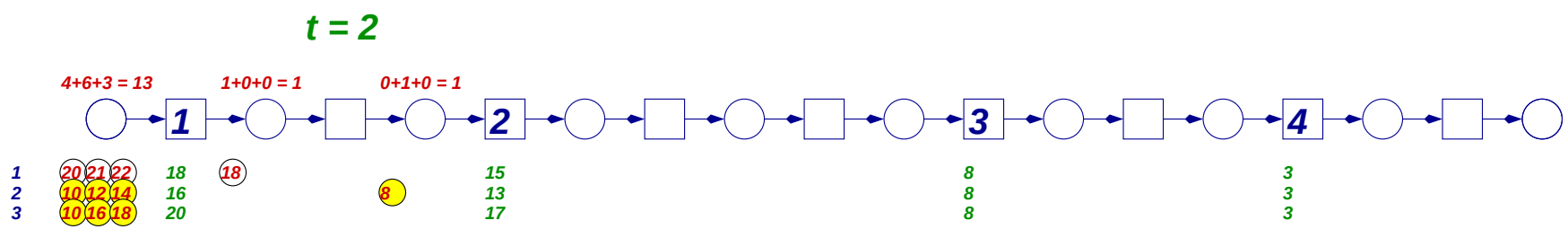
# CPP Example
## Graphical notation



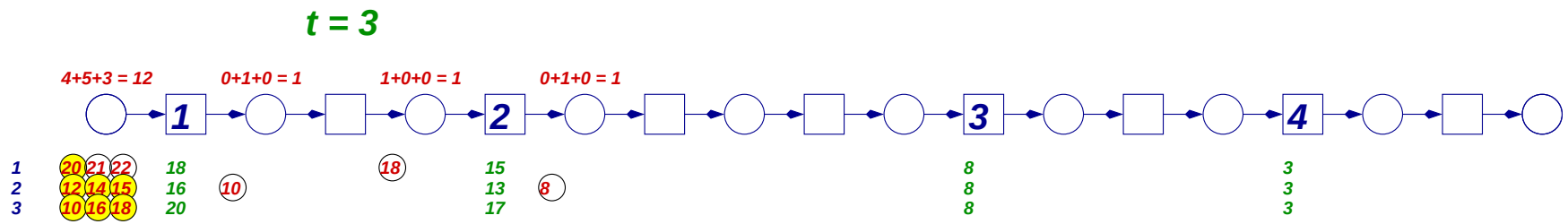
# CPP Example

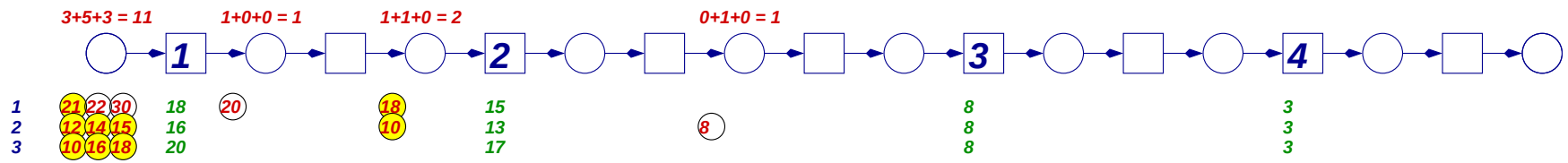


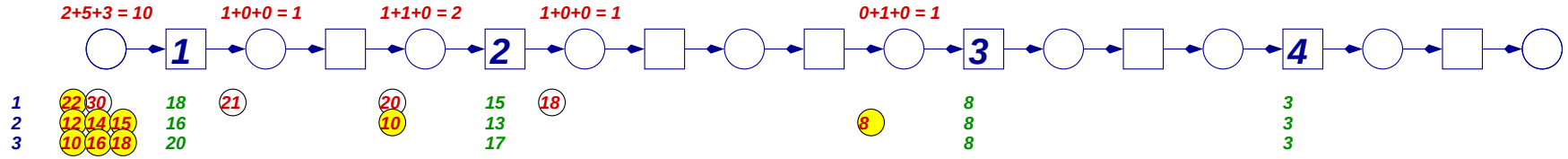



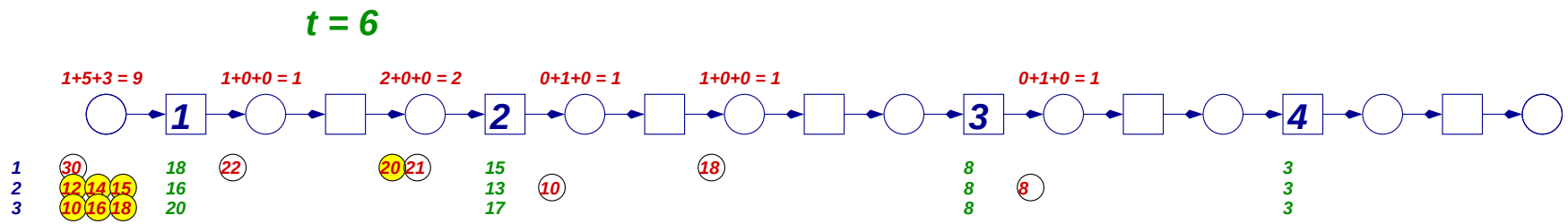
# CPP Example

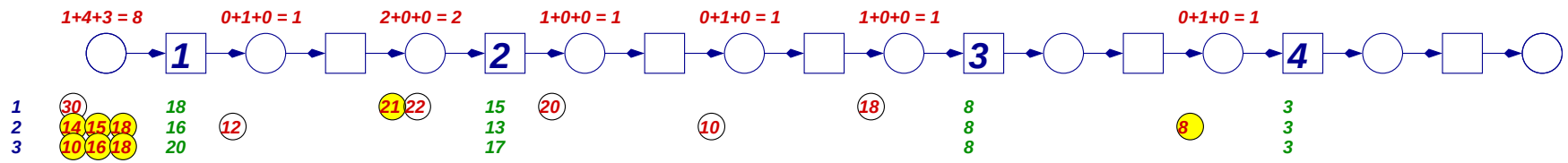


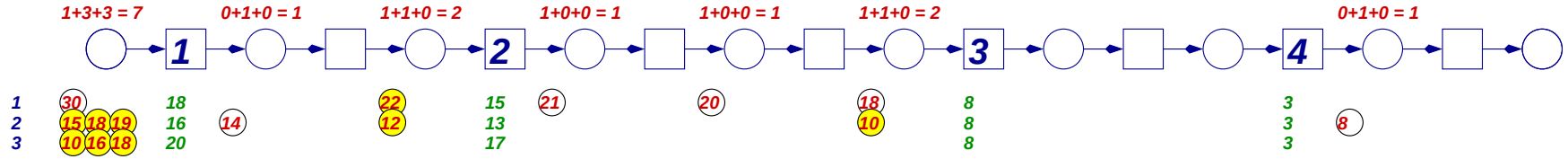



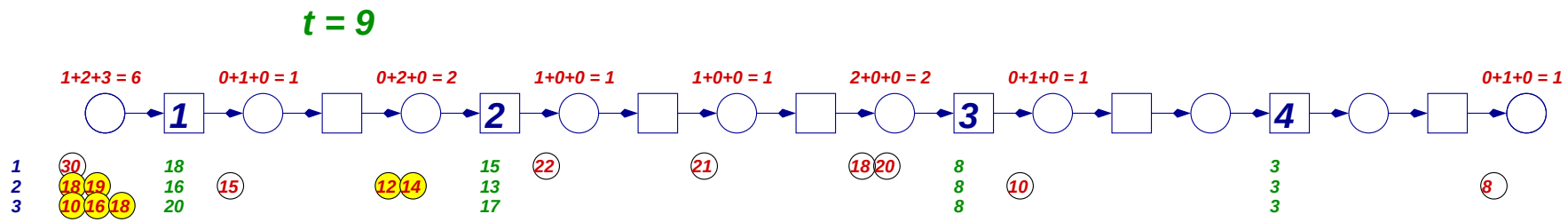
# CPP Example

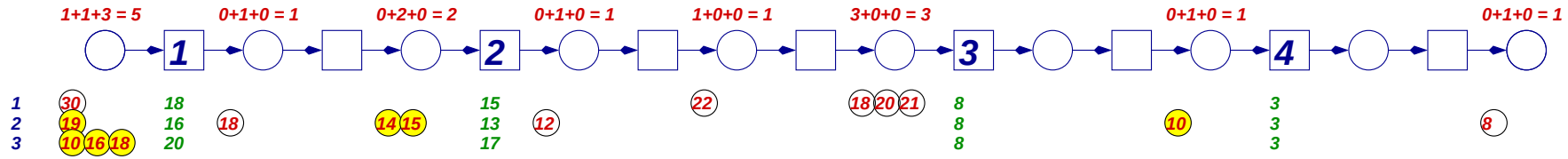


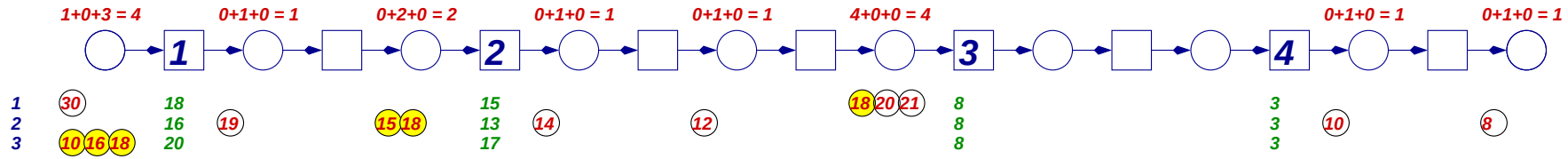



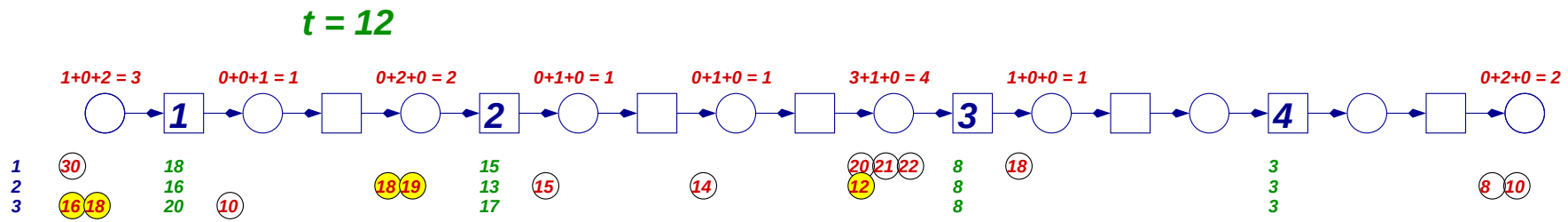
# CPP Example

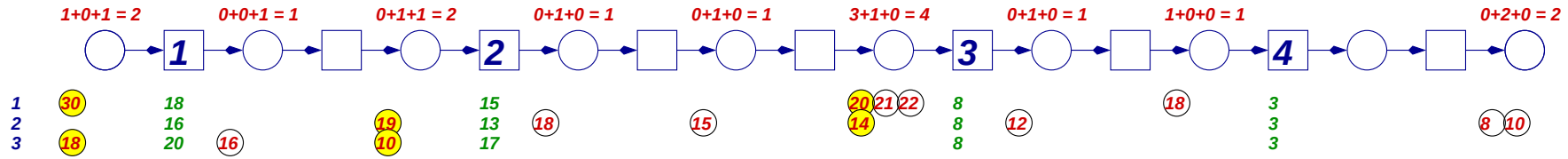


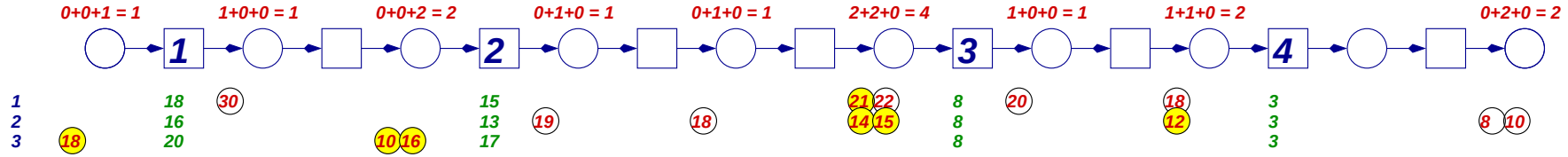



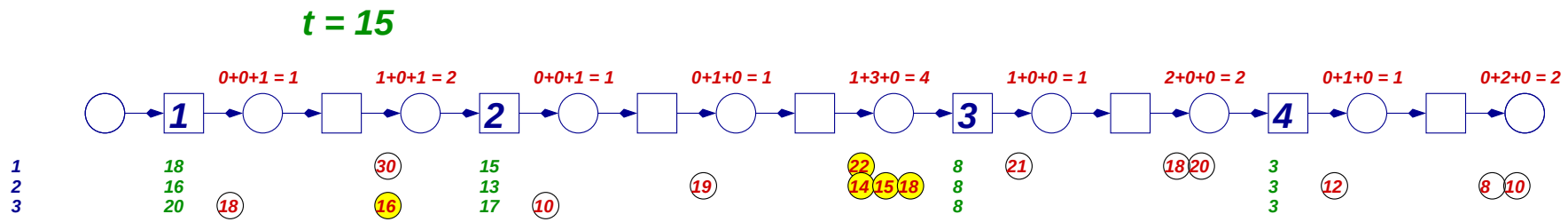
# CPP Example

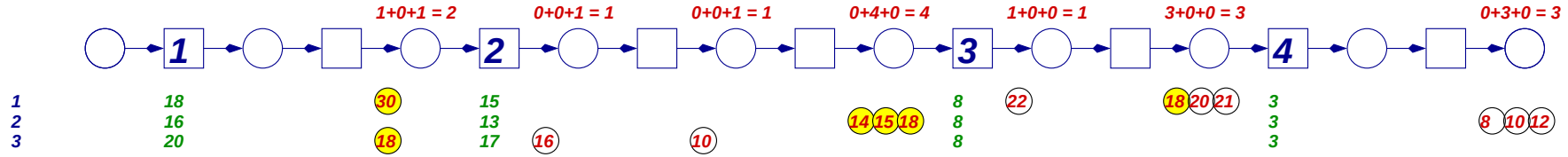


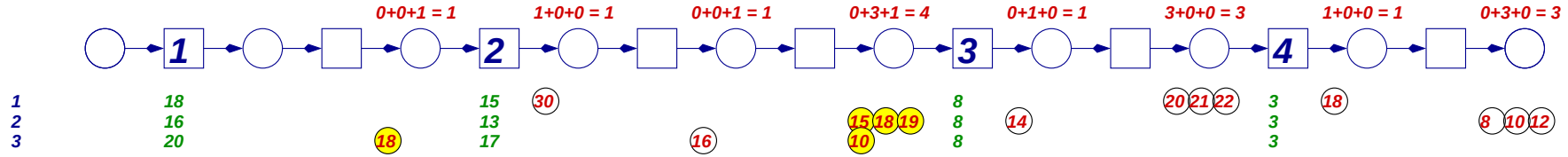



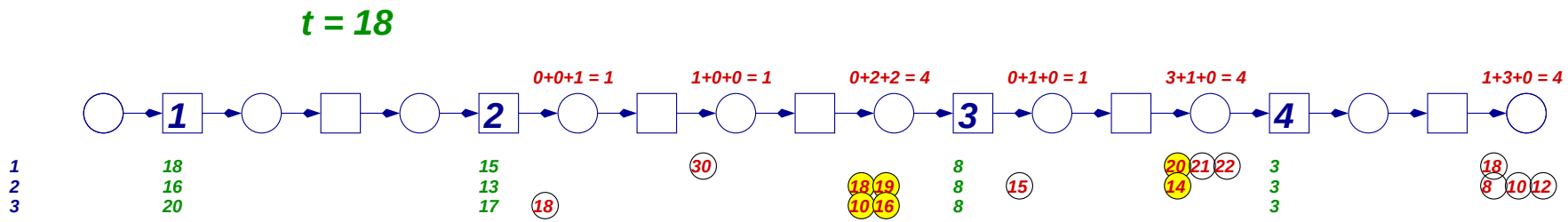
# CPP Example

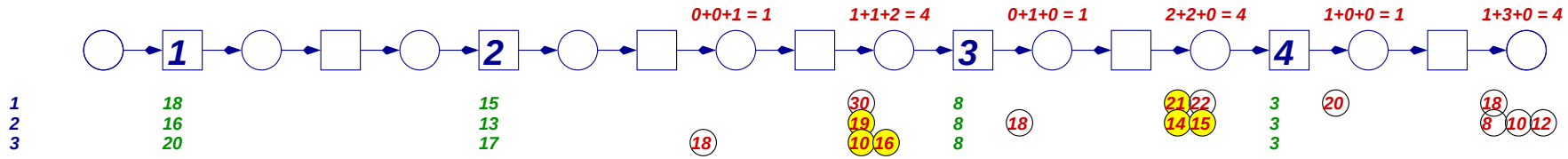


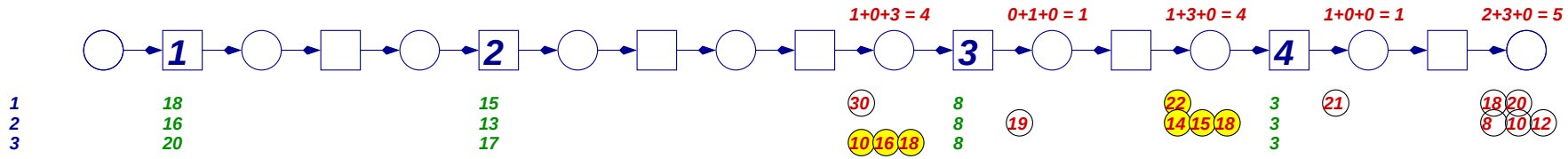



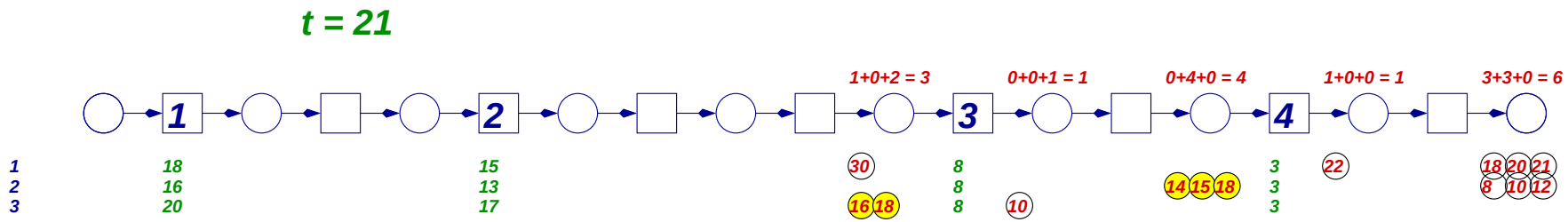
# CPP Example

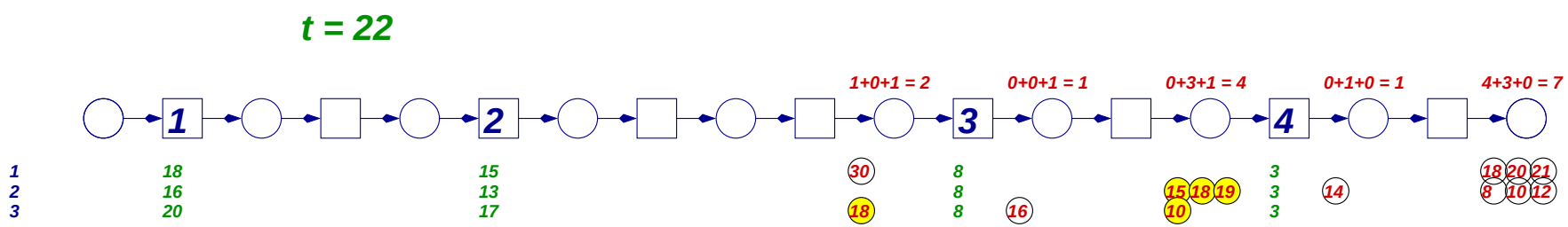


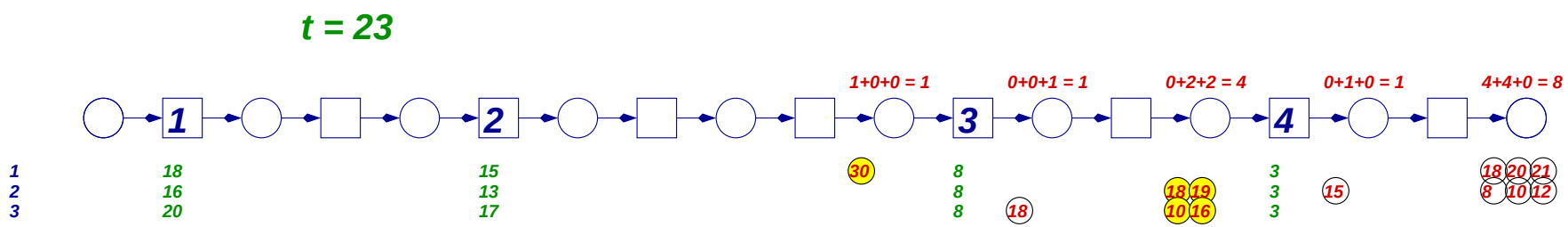



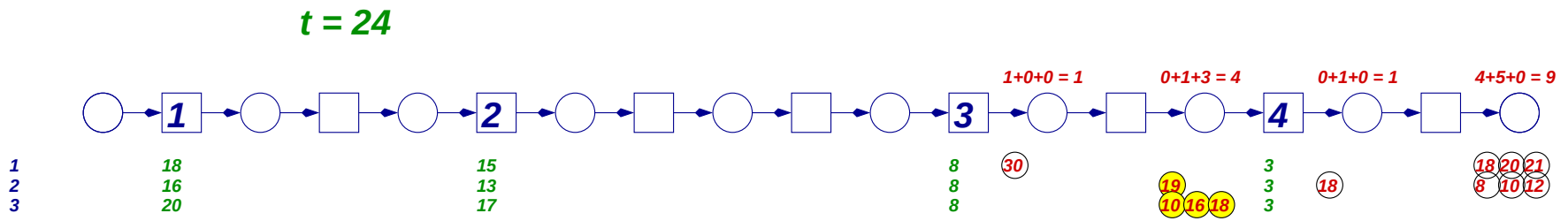
# CPP Example

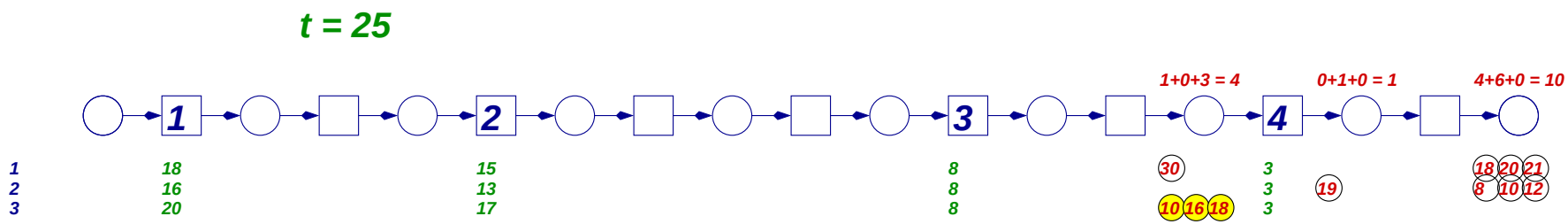


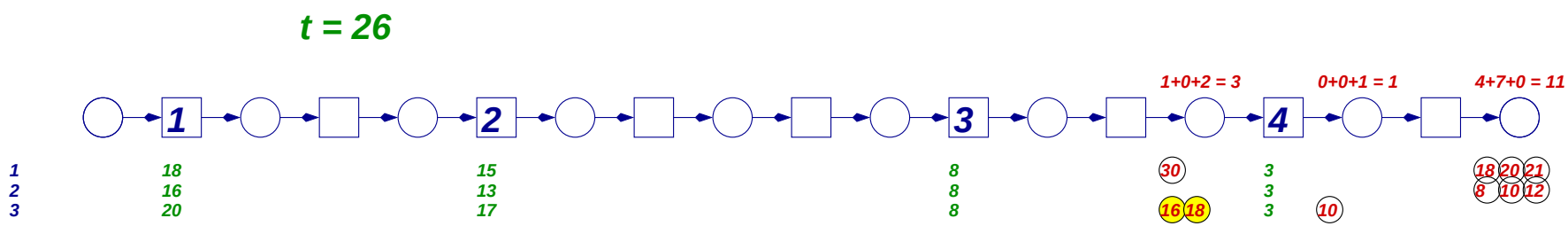



# CPP Example



# CPP Example