Java is a trademark of Sun Microsystems, Inc.

Building a Java™ Technology-Based Automation Controller: What, Why, How

Greg Bollella
Sun Microsystems

Outline

- Programmable Logic Controllers (PLC)
- Why we think they need to be replaced
- Fundamental differences between PLCs and PCs
- Not your Grandfather's Solaris
- The Sun Java Real-Time System (arguments)
- RTComms (Ethernet on steroids)
- Fieldbus support
- Health metric APIs
- The Sun Java Automation Controller

Programmable Logic Controller

Dinosaurs of the controls industry

- Everyone know what a 'relay' is?
- Think of a bank of relays with somewhat arbitrary connections and a bit of Boolean logic along the connections
- Well, a PLC is a digital version of the above

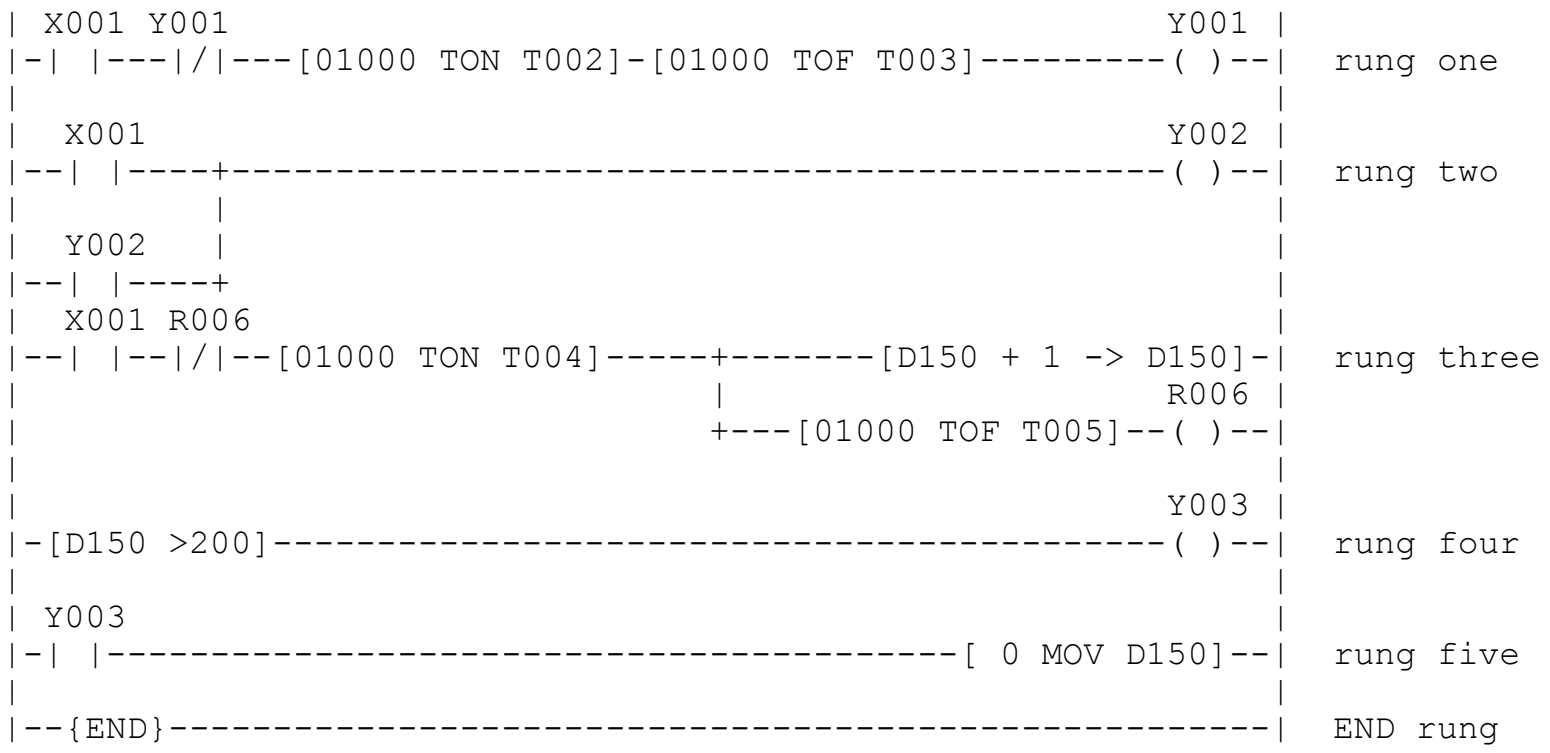
Wikipedia

A **programmable logic controller** (**PLC**) or **programmable controller** is a digital computer used for automation of electromechanical processes, such as control of machinery on factory assembly lines, amusement rides, or lighting fixtures.....

Early PLCs were designed to replace relay logic systems. These PLCs were programmed in "ladder logic", which strongly resembles a schematic diagram of relay logic. Modern PLCs can be programmed in a variety of ways, from ladder logic to more traditional programming languages such as BASIC and C. Another method is State Logic, a Very High Level Programming Language designed to program PLCs based on State Transition Diagrams.

http://en.wikipedia.org/wiki/Programmable_logic_controller

Ladder Logic



C for PLCs

- More modern PLCs allow the developer to write in C
- However,
  - It's a pretty limited set of the C language
  - Typically proprietary compiler and libs
- Not conducive to large complex code bases

What's wrong with the Status Quo

- Greenhouse gases
- Fossil fuel costs
- Road rage
- Elevator wait time
- Commute time
- PLC costs (initial and lock-in issues)
- Network isolation
- Come on, really, let's just get with it factor ...


http://www.freefoto.com

Differences between a PLC and a PC

- Real-time
- I/O
- Reliability
- Availability

Differences: Real-Time (1)

- Lots more later
- Controllers 'control' physical things
  - cf Computers manipulate virtual things
- Physical things have some interesting properties
  - Mass, inertia, temperature, velocity, wear, etc.
- Cannot actually change the laws of physics ...
  - cf Computer games
- The execution of logic is subservient to the physical parts of the system (not visa versa), thus ...

Differences: Real-Time (2)

- Logic must execute, with very little variation, **when** the physical system requires and needs it to execute
  - No time for Blue Screen of Death
  - No time for malloc to find some free memory
  - No time for garbage collection
  - No time for SMI
  - No time for OS housekeeping
- These are cyber-physical systems and inherently real-time

Differences: I/O

- Control is 50 % I/O
- General purpose I/O
- Analog and Digital
- These are the 'points' to which the sensors and actuators are connected to the controller
- I/O Slaves are small micro-controllers in themselves
- Communicate with higher level controllers over real-time communications channels typically called field buses

Differences: Reliability

- The IEEE definition: “... the ability of a system or component to perform its required functions under stated conditions for a specified period of time
- OK, think of your PC ... nuff said
- But, control systems operate in a much more restricted environment ....

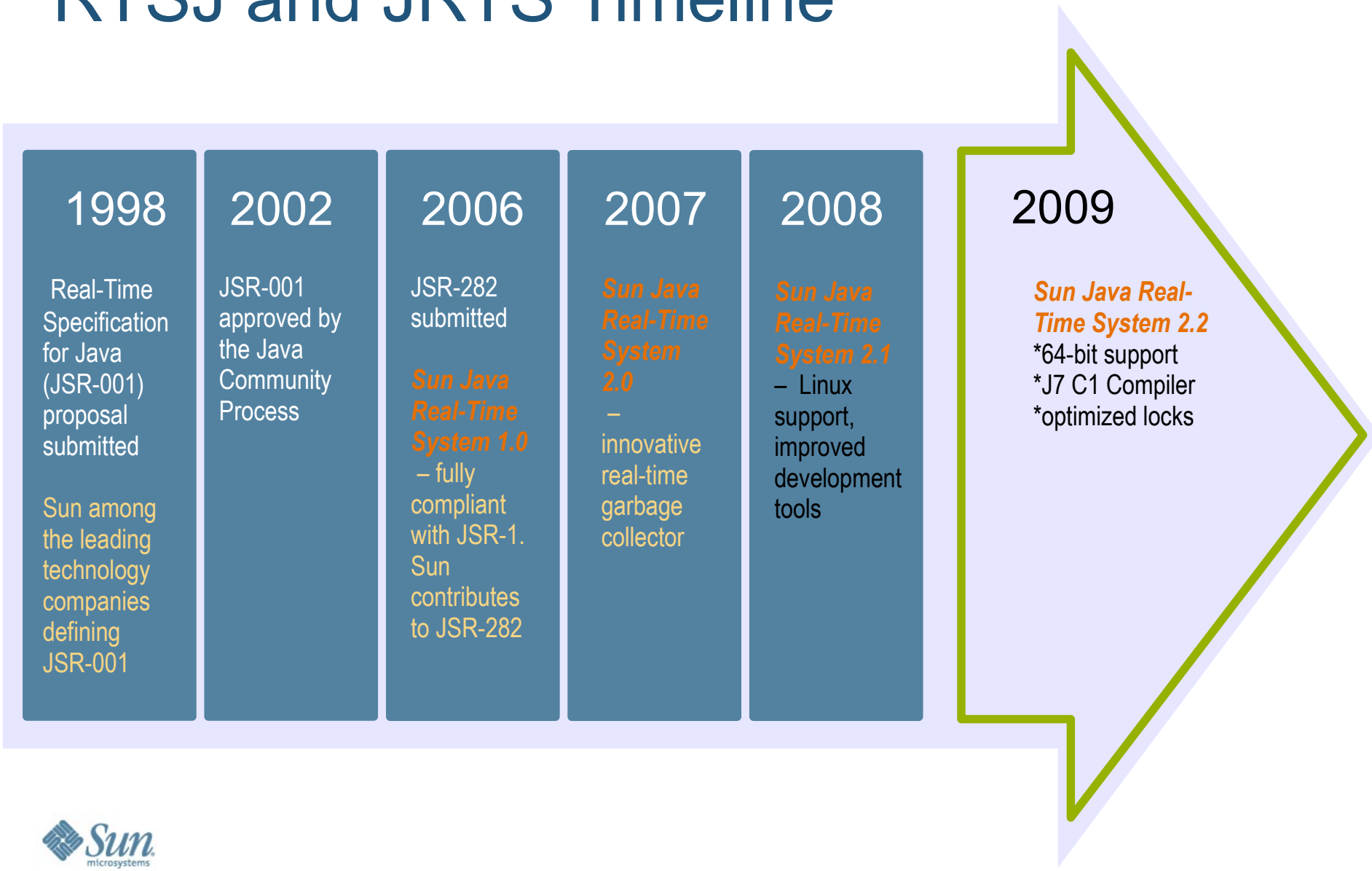
Differences: Availability

- The degree to which a system, subsystem, or equipment is operable and in a committable state at the start of a mission, when the mission is called for at an unknown, i.e., a random, time. Simply put, availability is the proportion of time a system is in a function condition
- E.g., we test a system for 5 days, so, over a 5 day period it has 100% reliability.
- Based on component MTBF we can compute an availability over some longer time period

Why Solaris?

- A major value of the SJAC is to bring modern computing environments to the control environment
- JavaRTS + Solaris provides an excellent real-time application execution environment as well as an excellent non-real-time application execution environment
- Solaris provides, to the controller infrastructure all the benefits it provides to the IT data center

RTSJ and JRTS Timeline



Core Areas of the RTSJ

- Scheduling and Dispatching
- Synchronization
  - Priority inversion avoidance
- Memory Management
- Asynchronous events and responses
- Time, Clocks and Timers
- Asynchronous Transfer of Control
- System functions
  - Signal interaction

Schedulable Objects

- Notion of a `Schedulable` object
  - Interface implemented by executable entities
  - Execution managed by a `Scheduler`
- RTSJ semantics are defined only for pre-defined types of `Schedulable` objects, referred to as "*schedulable objects*" (SO)
  - Instances of `RealtimeThread` and its subclasses (RTTs
  - Instances of `AsyncEventHandler` and its subclasses (AEHs)
- A plain `java.lang.Thread` (JLT) is not a SO

Real-Time Threads

- Class `RealtimeThread`
  - Extends `java.lang.Thread`
  - Implements `Schedulable`
- Defines static methods to operate on the current real-time thread:
  - `boolean waitForNextPeriod()`

No-Heap Real-Time Threads

- Class `NoHeapRealtimeThread`
  - Extends `RealtimeThread`
- Forbidden from accessing any object allocated in the Java heap
  - Always executes in either *scoped* or *immortal* memory
- Can be immune to interference from the GC, provided

Release Characteristics

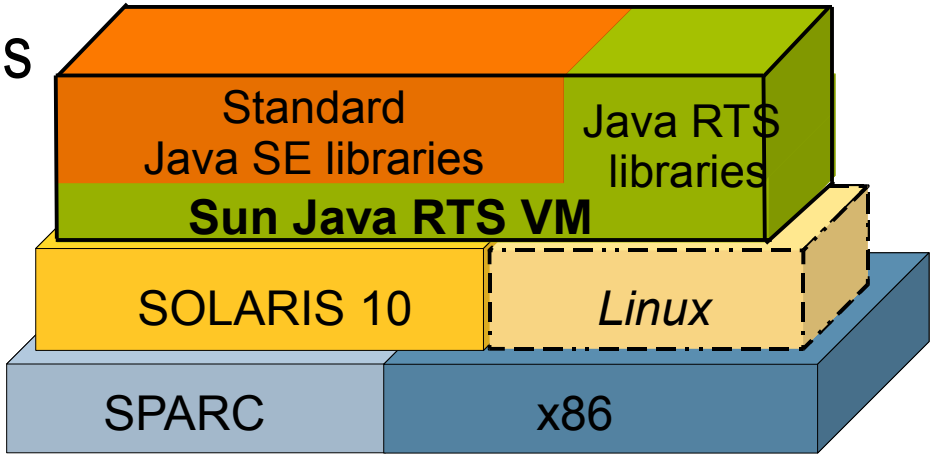
- Scheduling theory characterises tasks by their pattern of *release*
  - i.e. when, and how often, the task is ready for execution
- A release is triggered by some event and defines an *arrival* of the SO
- A release *completes* when the SO "stops executing"
- Periodic: released regularly based on a set *period*
- Aperiodic: release pattern is unknown
- Sporadic: release pattern is unknown, as for aperiodic, except that the *minimum inter-arrival time* (MIT) is known

The Sun Java Real-Time System

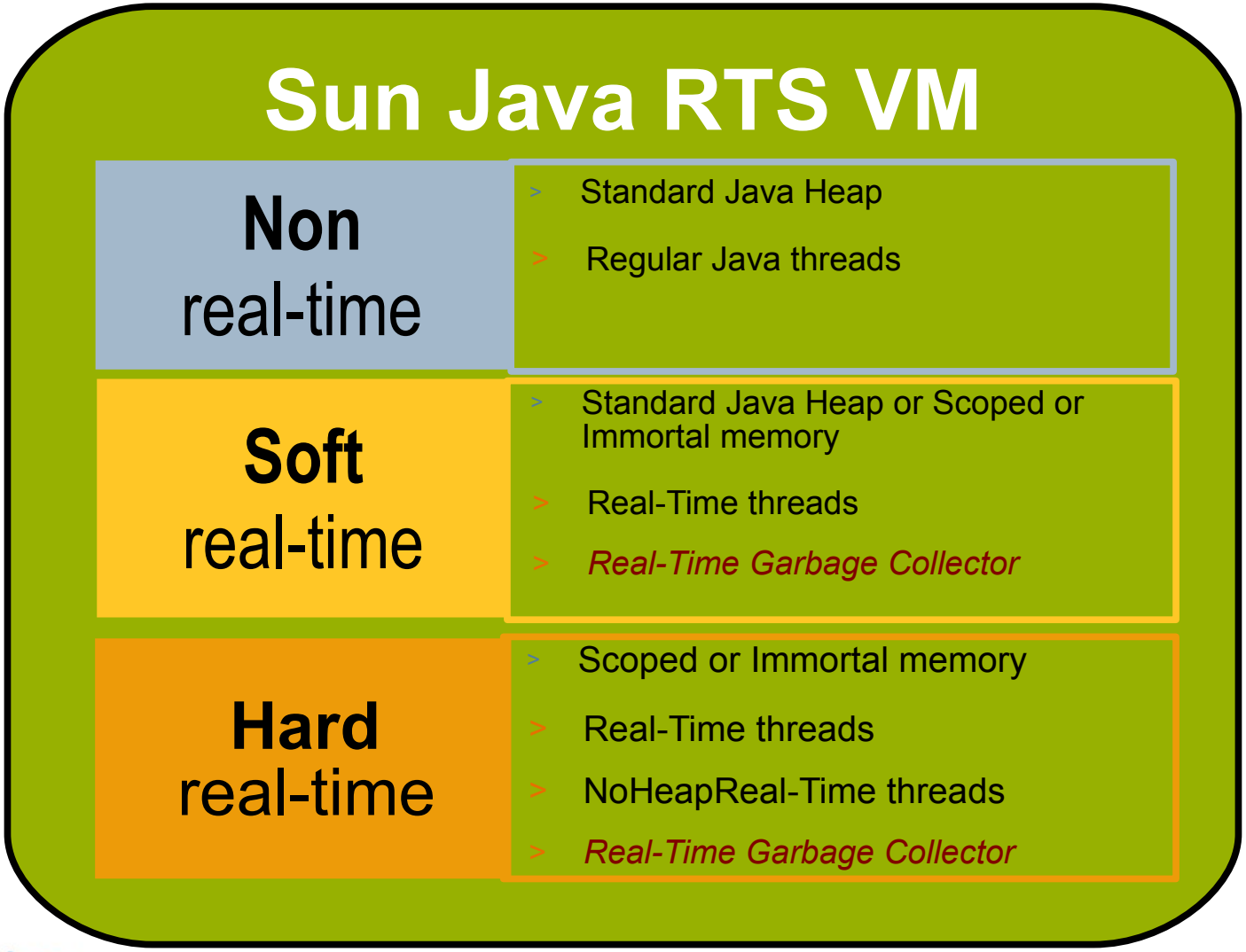
- An implementation of the RTSJ
- Product available since 2006
- Java SE compliant
- JSR-01 compliant
- Available for over 800 h/w platforms
- Maximum latencies of 15 microseconds possible
- Full real-time garbage collection

Sun's Java Real-Time System – absolute execution predictability with all the benefits of the Java platform

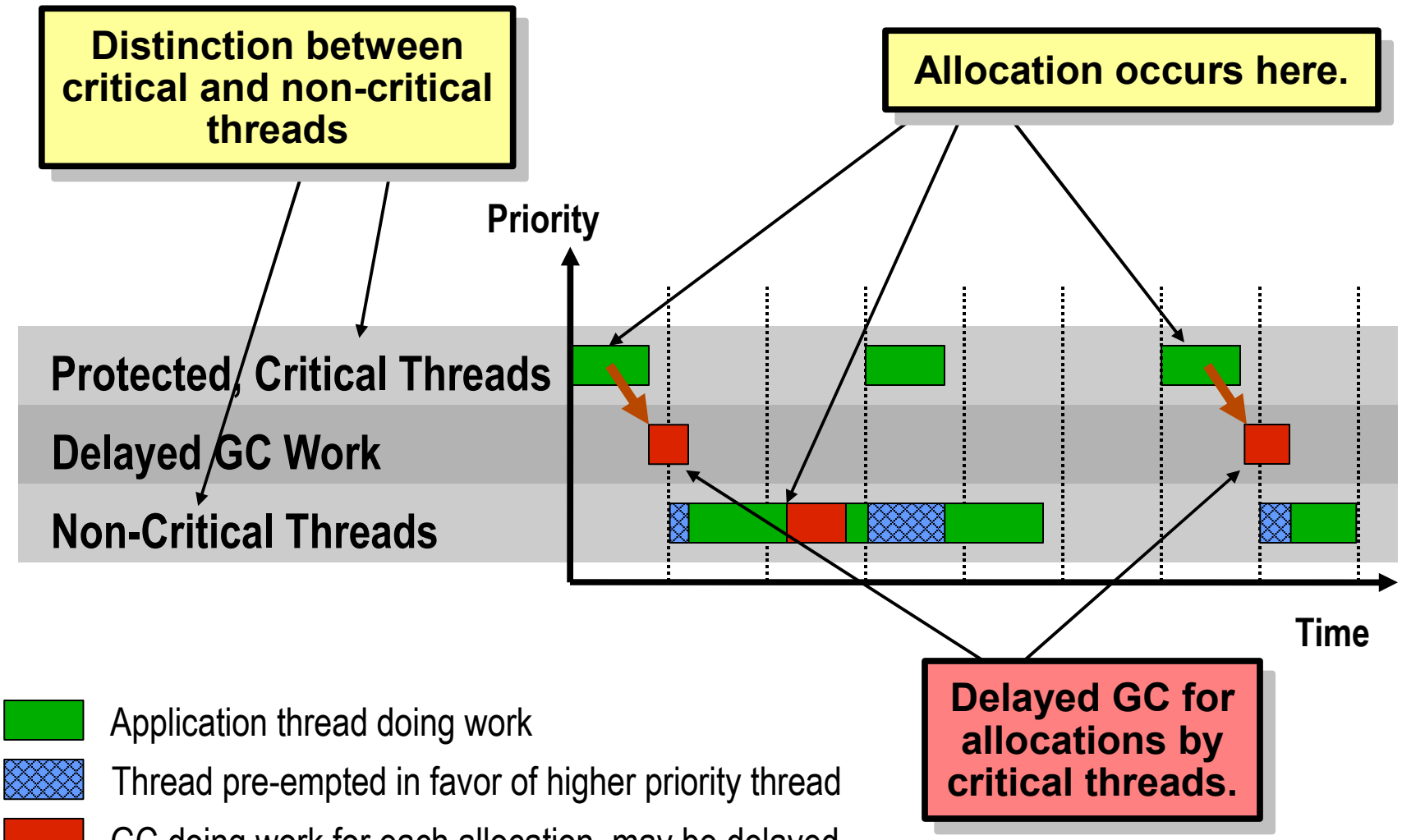
- **Truly predictable**
  - Formal mechanisms to eliminate latency issues
- **Open**
  - Based on open, community-driven standards



Overall System Model



Example: Henriksson's GC



The Arguments

- The Java Platform provides tools (development and debug), libraries, and an environment appropriate for easily developing complex control algorithms
- The RTSJ provides sufficiently high-level real-time development abstractions
- The RTSJ abstracts timing details
- Exactly the same code can run in a simulation as well as the target system
- Graphical interface and control code execute w/in the same JVM

The Arguments

- The Java Platform provides tools (development and debug), libraries, and an environment appropriate for easily developing complex control algorithms
- The RTSJ provides sufficiently high-level real-time development abstractions
- The RTSJ abstracts timing details
- Exactly the same code can run in a simulation as well as the target system
- Graphical interface and control code execute w/in the same JVM

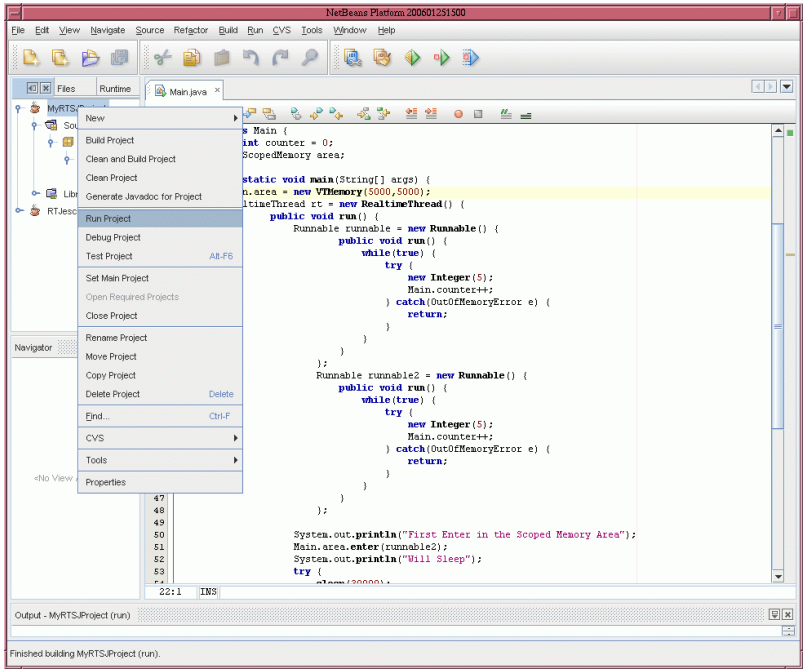
The Java Platform

- Vibrant Community
  - 6M developers
- Tools, IDEs, debug, profiling, real-time
- Libraries, thousands
- Google what you need to know
- Ubiquitous byte code (runs anywhere)

NetBeans IDE Support

Download the Java RTS plugin, and

- cross-develop on the host
- write ALL code in Java
- deploy over the network
- execute on the target



The Arguments

- The Java Platform provides tools (development and debug), libraries, and an environment appropriate for easily developing complex control algorithms
- The RTSJ provides sufficiently high-level real-time development abstractions
- The RTSJ abstracts timing details
- Exactly the same code can run in a simulation as well as the target system
- Graphical interface and control code execute w/in the same JVM

High Level RT Development

- Release characteristics
  - periodic, aperiodic, sporadic
- Deadline Miss and Cost Overrun Handlers
- Periodic and One-Shot Timers
- AsyncEvent and AsyncEventHandler
- Real-Time Garbage Collection
- Precise Dispatch Semantics

The Arguments

- The Java Platform provides tools (development and debug), libraries, and an environment appropriate for easily developing complex control algorithms
- The RTSJ provides sufficiently high-level real-time development abstractions
- The RTSJ abstracts timing details
- Exactly the same code can run in a simulation as well as the target system
- Graphical interface and control code execute w/in the same JVM

Abstraction of Timing

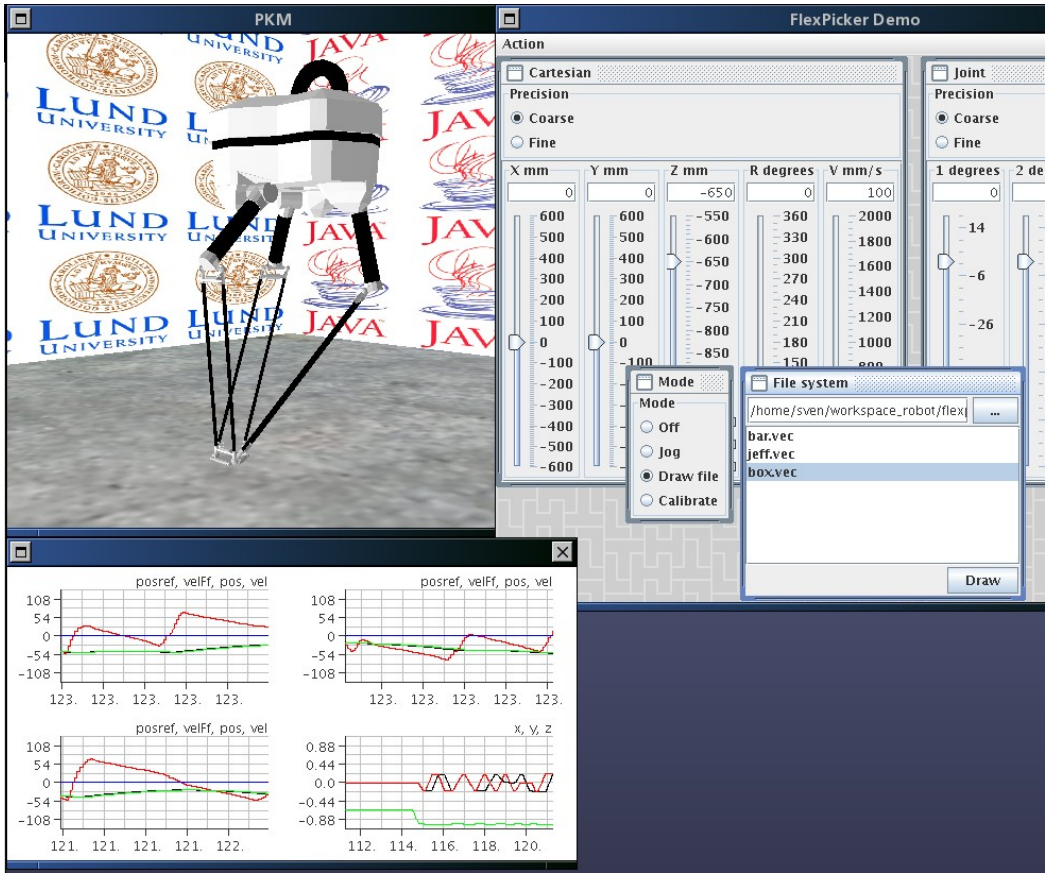
```
while (condition) {
    readSensors();
    computeControl();
    sendCommands();
    waitForNextPeriod();
}
```

The Arguments

- The Java Platform provides tools (development and debug), libraries, and an environment appropriate for easily developing complex control algorithms
- The RTSJ provides sufficiently high-level real-time development abstractions
- The RTSJ abstracts timing details
- Exactly the same code can run in a simulation as well as the target system
- Graphical interface and control code execute w/in the same JVM

Simulation – Target; Same Code

- Java3D visualization
- Simple dynamics simulation
- Runs on J2SE
  - ***Exactly*** the same control code as for the physical robot
  - Debugging on the desktop



The Arguments

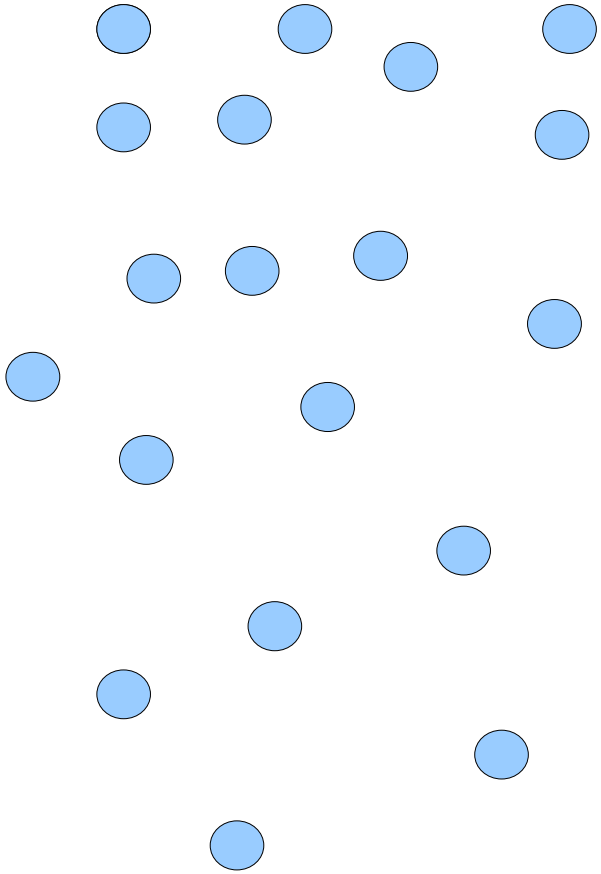
- The Java Platform provides tools (development and debug), libraries, and an environment appropriate for easily developing complex control algorithms
- The RTSJ provides sufficiently high-level real-time development abstractions
- The RTSJ abstracts timing details
- Exactly the same code can run in a simulation as well as the target system
- Graphical interface and control code execute w/in the same JVM

Graphical Interface + Control

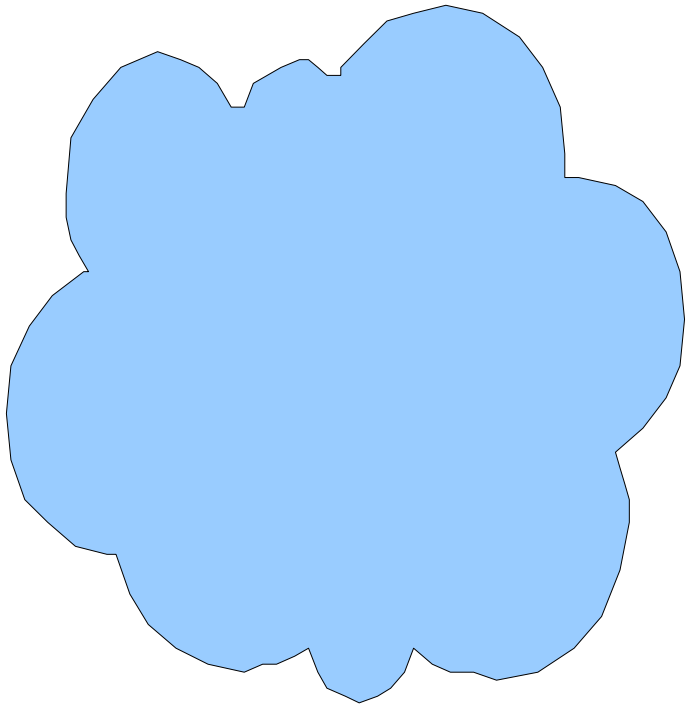
- Project Sydney
- BlueWonder System
- ProfiBus
- Industrial Automation Sensors and Actuators
- GUI and Control in Two Weeks

Integration (Yesterday)

Factory Floor

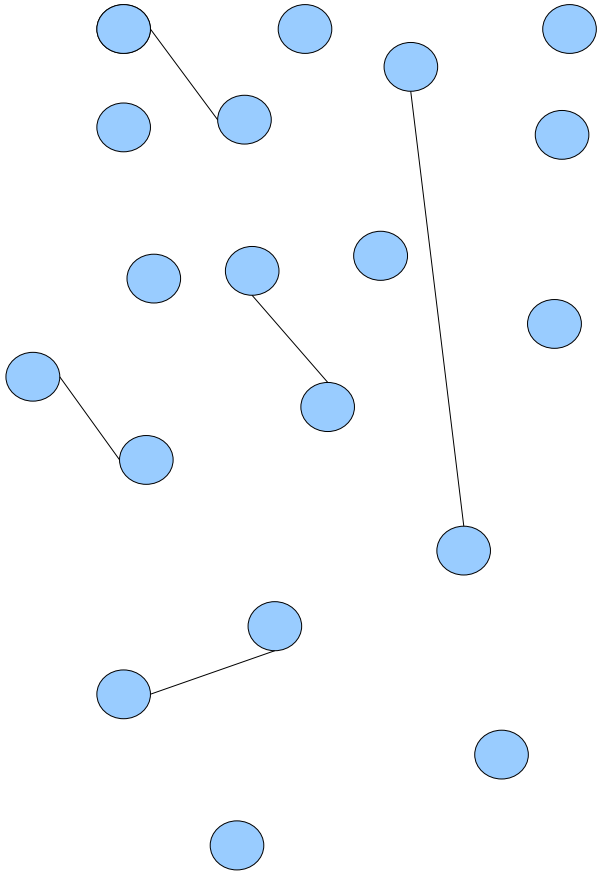


Organization Intranet

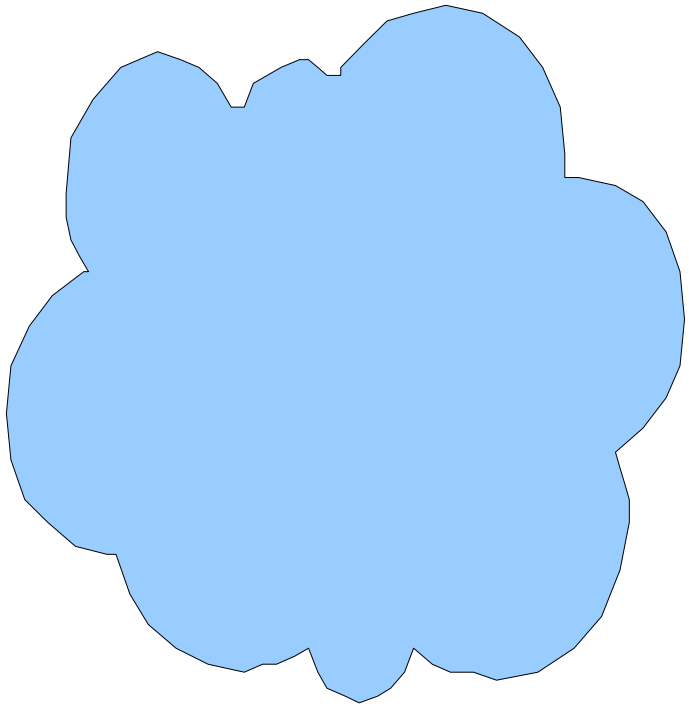


Integration (Yesterday)

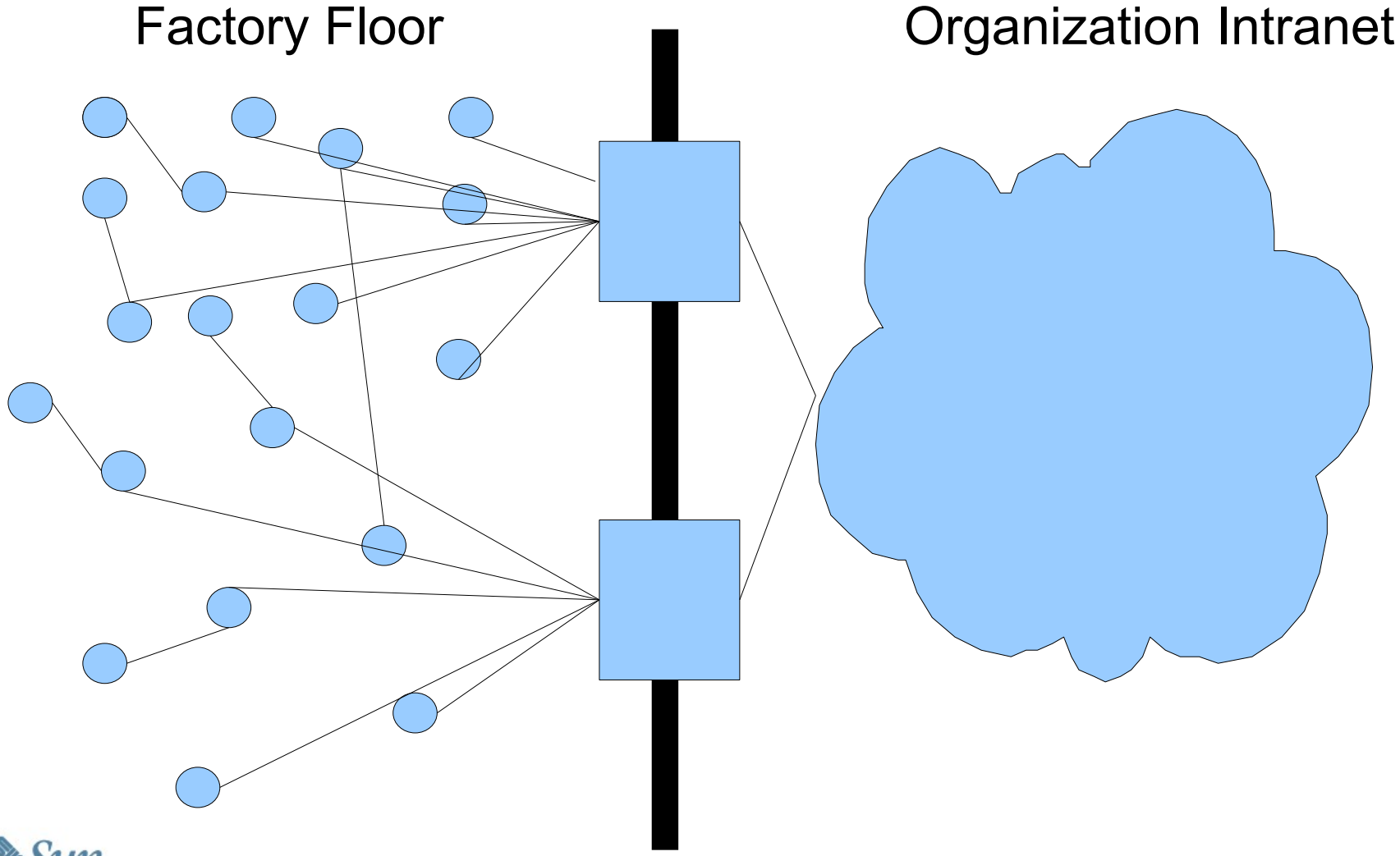
Factory Floor



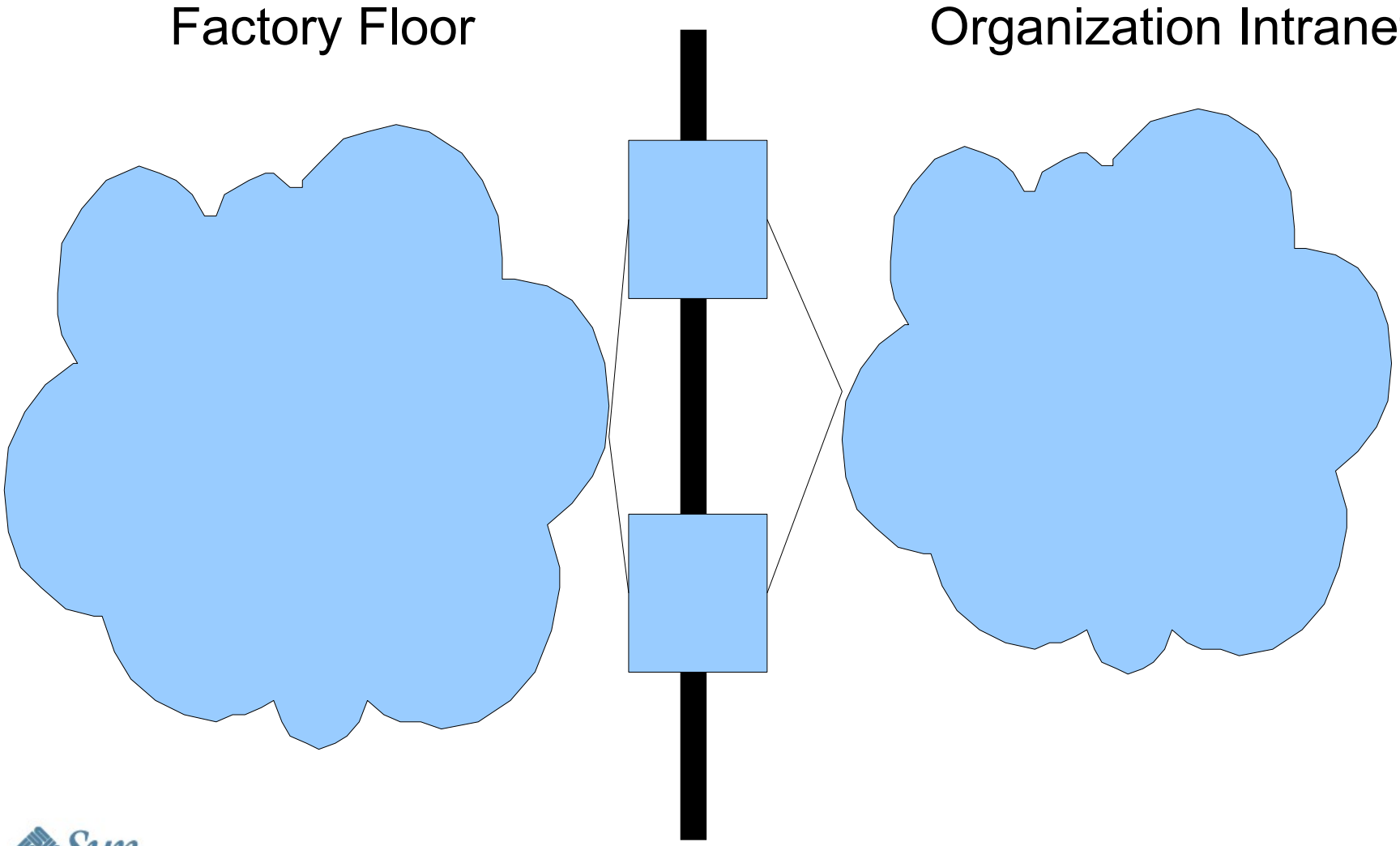
Organization Intranet



Integration (Today)



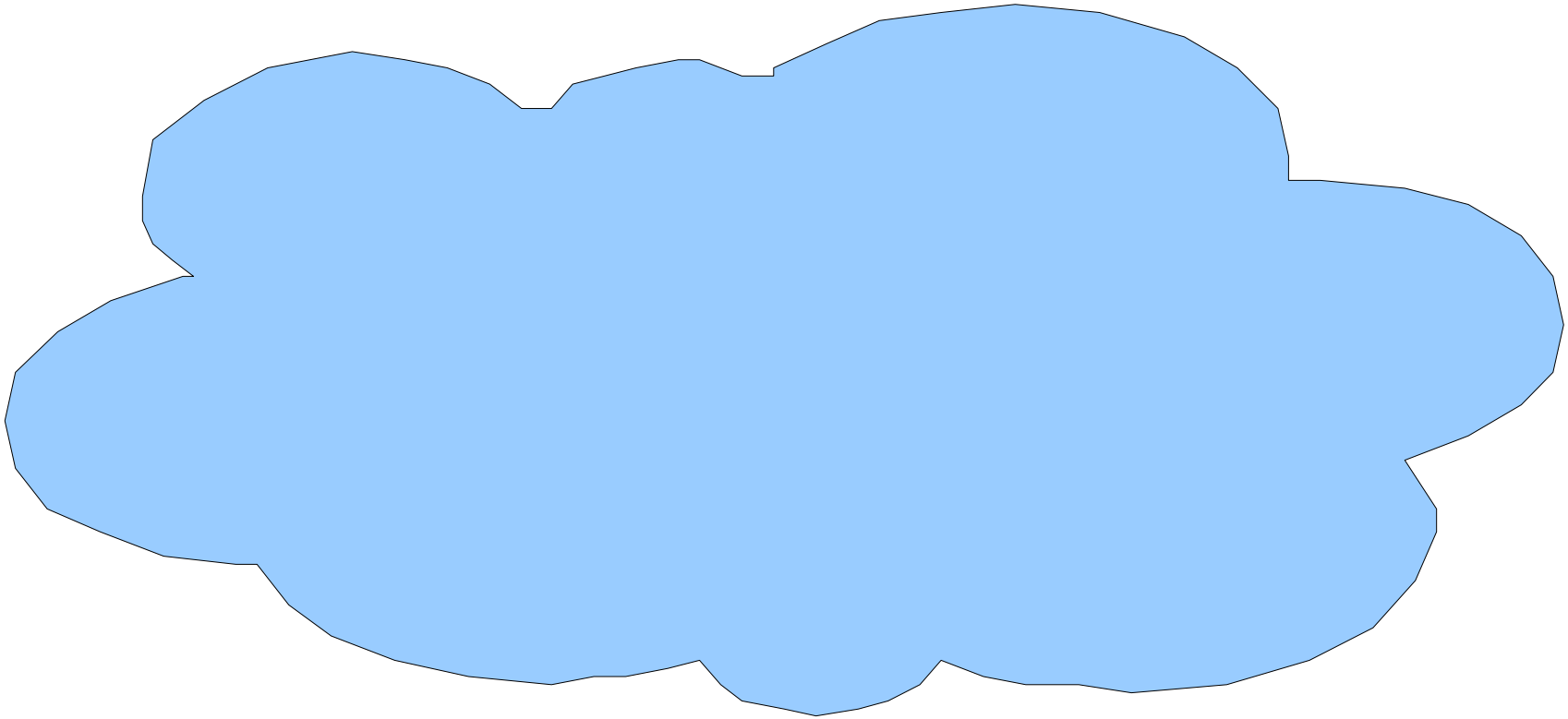
Integration (Tomorrow / Today)



Integration (Today with BlueWonder)

Factory Floor

Organization Intranet



RTComms

- Real-time communication over Ethernet
- Application <--> DD <--> e1000g
- Point-to-point
- Synchronous protocol
- Packet corruption handled
- Redundancy if two interfaces available

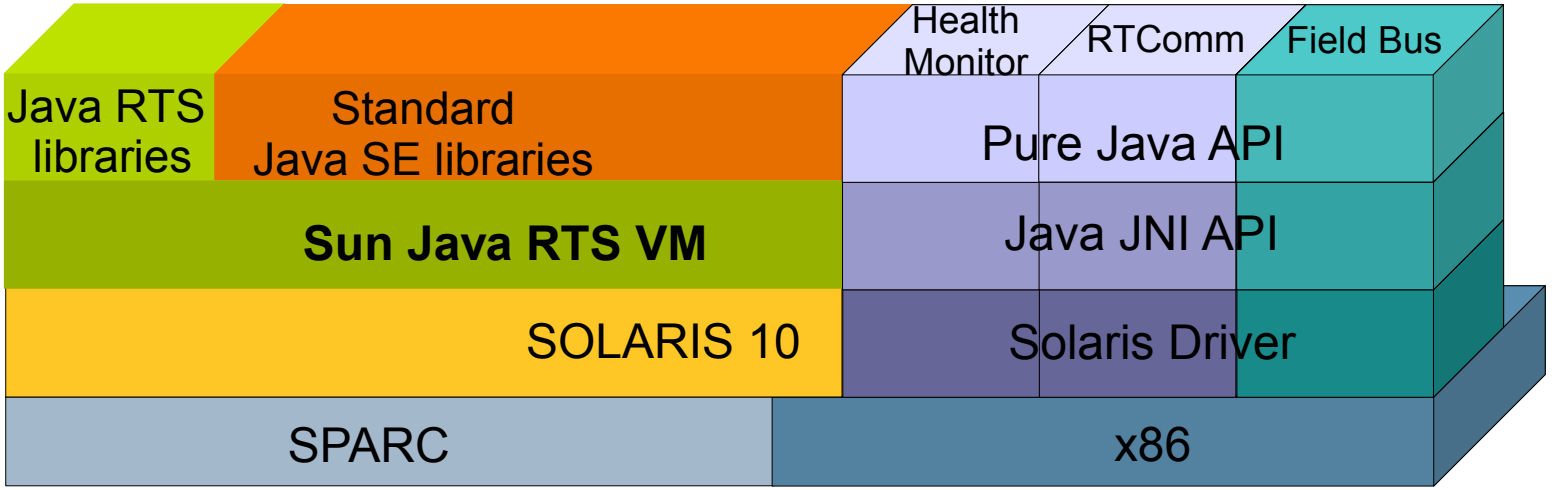
Fieldbus Support

- Industrial automation uses a variety of real-time, synchronous communications protocols to connect I/O slaves and controllers
- ProfiBus, ProfiNet, Modbus, DeviceNet, ...
- Support through PC104+ cards and Solaris device drivers

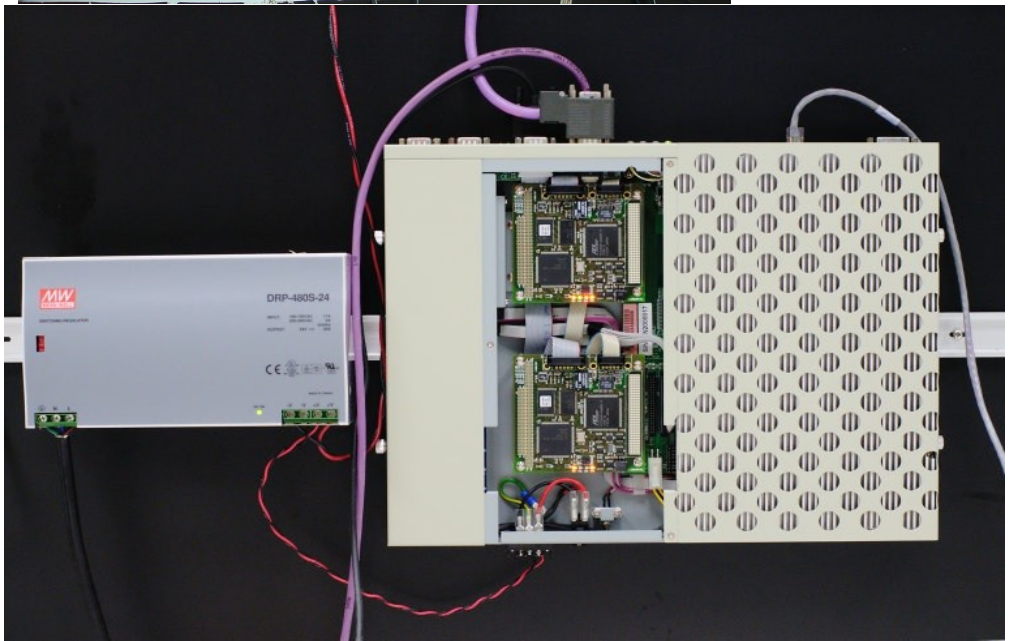
Health Metric APIs

- Java APIs and Hardware support for:
- CPU and system temp
- Power supply and battery voltages
- CPU utilization
- Memory utilization
- Network and disk utilization
- System serial number
- Easily extended for additional `kstat`

SJAC Additional APIs



Sun Java Automation Controller



Thank You

Greg Bollella
greg.bollella@sun.com