

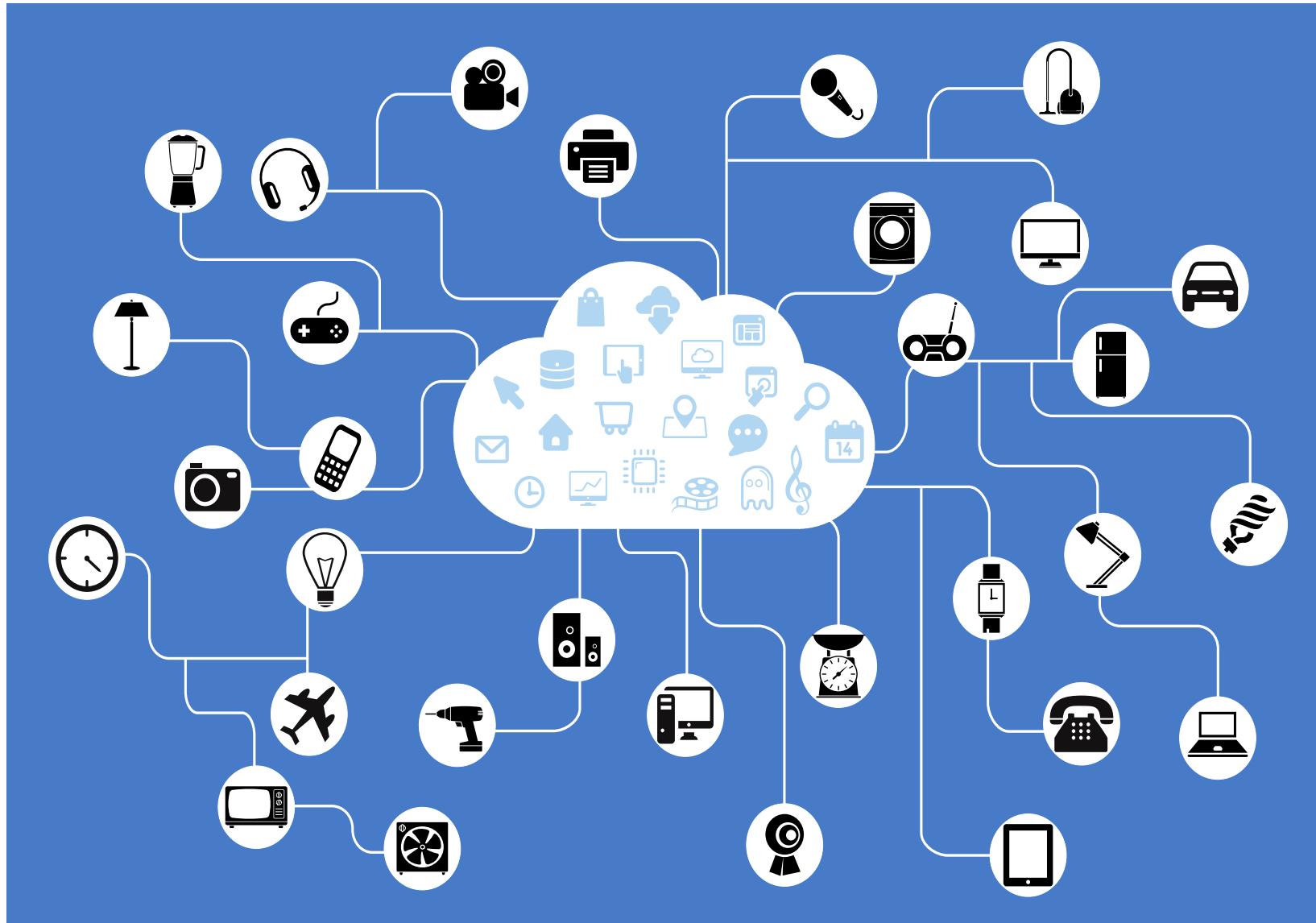
# Runtime monitoring for concurrent systems

Y. Yamagata, C. Artho, M. Hagiya, J. Inoue, Lei Ma,  
Y. Tanabe and M. Yamamoto

Sep. 29, RV2016

# IoT World

# Concurrent, Asynchronous, Distributed



# Our Goal

- Monitoring method for concurrent systems
- Support for asynchronous events
- Bottom-up approach for specification
- Solid theoretical foundation

# Our achievement

- CSP\_E, a simple extension of Hoare's CSP
- Formal semantics of CSP\_E
- Implement it as DSL in Scala
- It shows acceptable performance (speed and memory) for middle-scale (100-1000 processes) systems

# Related works

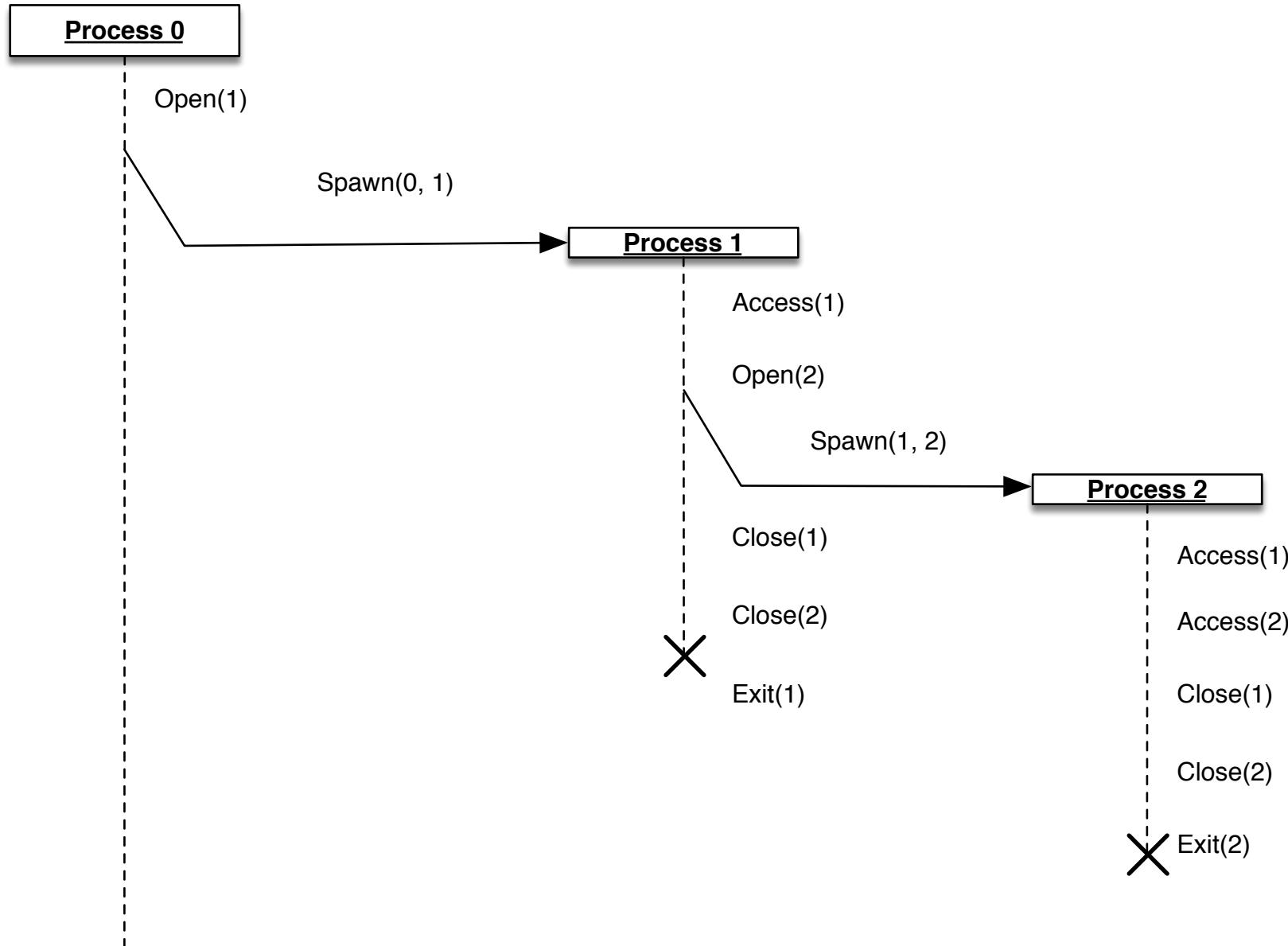
## Most of traditional RV

- Global properties
- Top-down
- Specification languages
  - Automata
  - Temporal logics
  - Formal grammars
  - Rule-based systems

## Our Approach

- Components and interaction
- Bottom-up
- Specification language
  - A process calculus (CSP)

# Unix-like processes and file descriptors



# Comparison to QEA, an automata based approach

## CSP\_E code

```
def process(pid: Int, openFiles: Set[Int]): Process = ?? {
    case Event('Spawn, `pid`, child_pid: Int) =>
        process(pid, openFiles) ||| process(child_pid, openFiles)
    case Event('Open, `pid`, fd: Int) if !openFiles(fd) =>
        process(pid, openFiles + fd)
    case Event('Access, `pid`, fd: Int) if openFiles(fd) =>
        process(pid, openFiles)
    case Event('Close, `pid`, fd: Int) if openFiles(fd) =>
        process(pid, openFiles - fd)
    case Event('Exit, `pid`) if pid != 0 && openFiles.isEmpty => SKIP
}

def uniqueProcess(pidSet : Set[Int]) : Process = ?? {
    case Event('Spawn, _, child_pid : Int) if ! pidSet(child_pid) =>
        uniqueProcess(pidSet + child_pid)
    case Event('Exit, pid : Int) if pidSet(pid) => uniqueProcess(pidSet - pid)
}

def system = process(0, Set.empty) || Set('Spawn, 'Exit) || uniqueProcess(Set(0))
```

# Comparison to QEA, an automata based approach

## QEA code

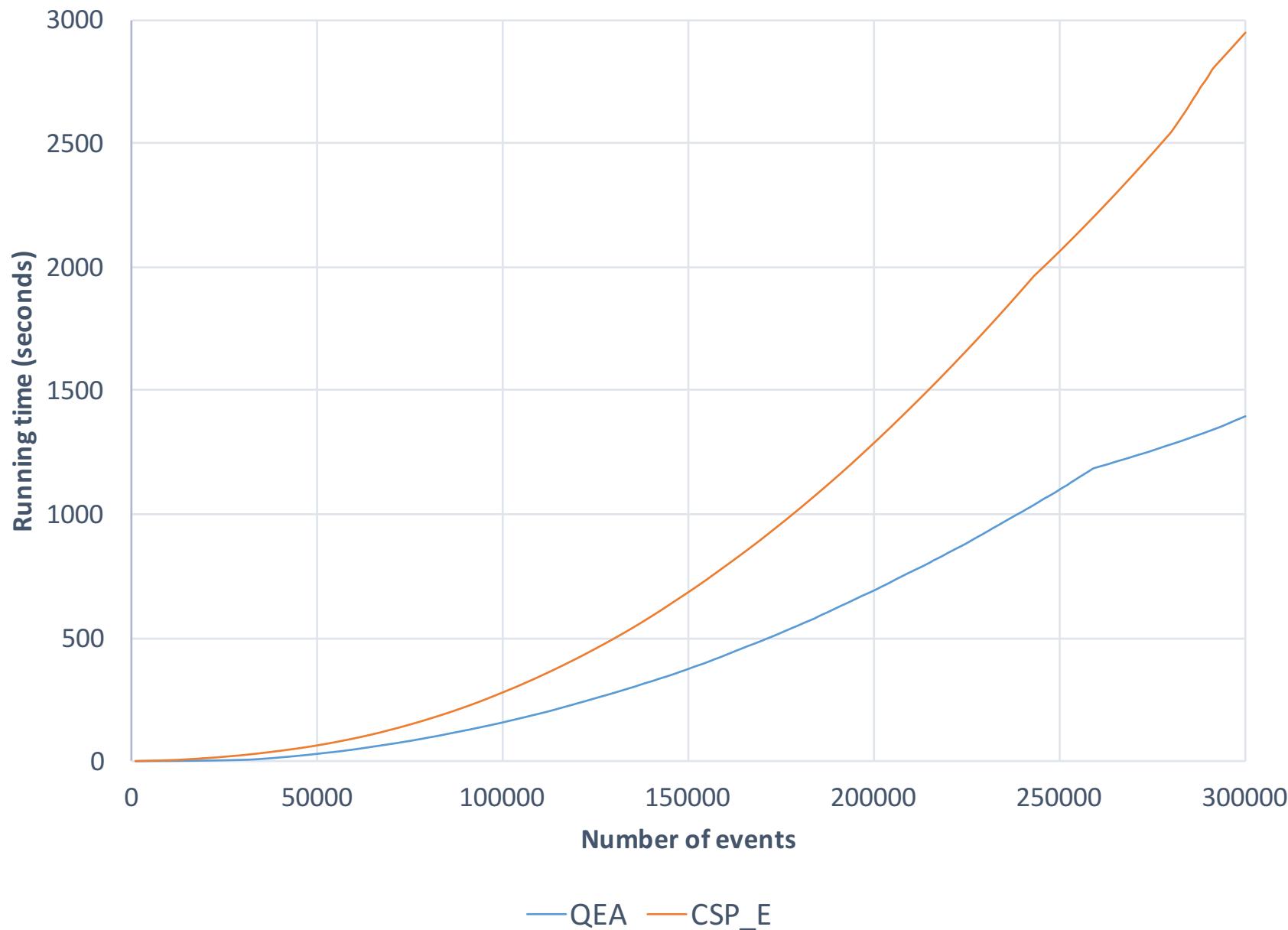
### File descriptor monitor

```
qea {
  Forall(fd)
  accept skip(init) {
    spawn(parent, child) if [ parent in PS ]
      do [ PS.add(child) ] -> init
    open(pid, fd) if [ pid in PS ] -> failure
    open(pid, fd) if [ not pid in PS ]
      do [ PS.add(pid) ]-> init
    access(pid, fd) if [ not pid in PS ] -> failure
    close(pid, fd) if [ pid in PS ]
      do [ PS.remove(pid) ] -> init
    close(pid, fd) if [not pid in PS ] -> failure
    exit(pid) if [ pid in PS ] -> failure
  }
}
```

### Process monitor

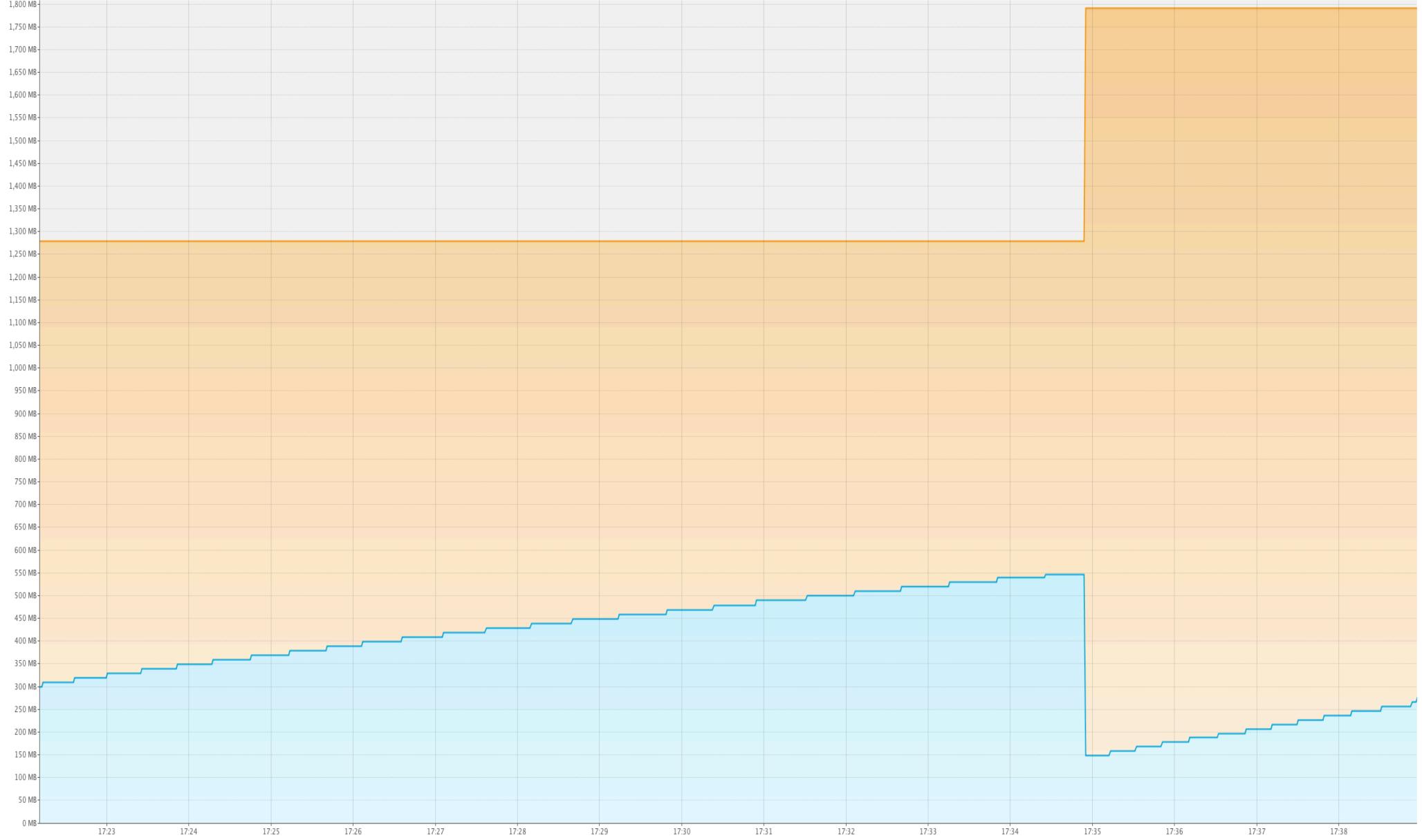
```
qea {
  accept next(init) {
    spawn(parent, child)
      if [ (parent in PS || parent = 0) &&
          not child in PS && child = 0]
        do [ PS.add(child) ] -> init
    exit(pid) if [ pid in PS ]
      do [ PS.remove(pid) ] -> init
    open(pid, fd) -> init
    access(pid, fd) -> init
    close(pid, fd) -> init
  }
}
```

## Comparison of running time



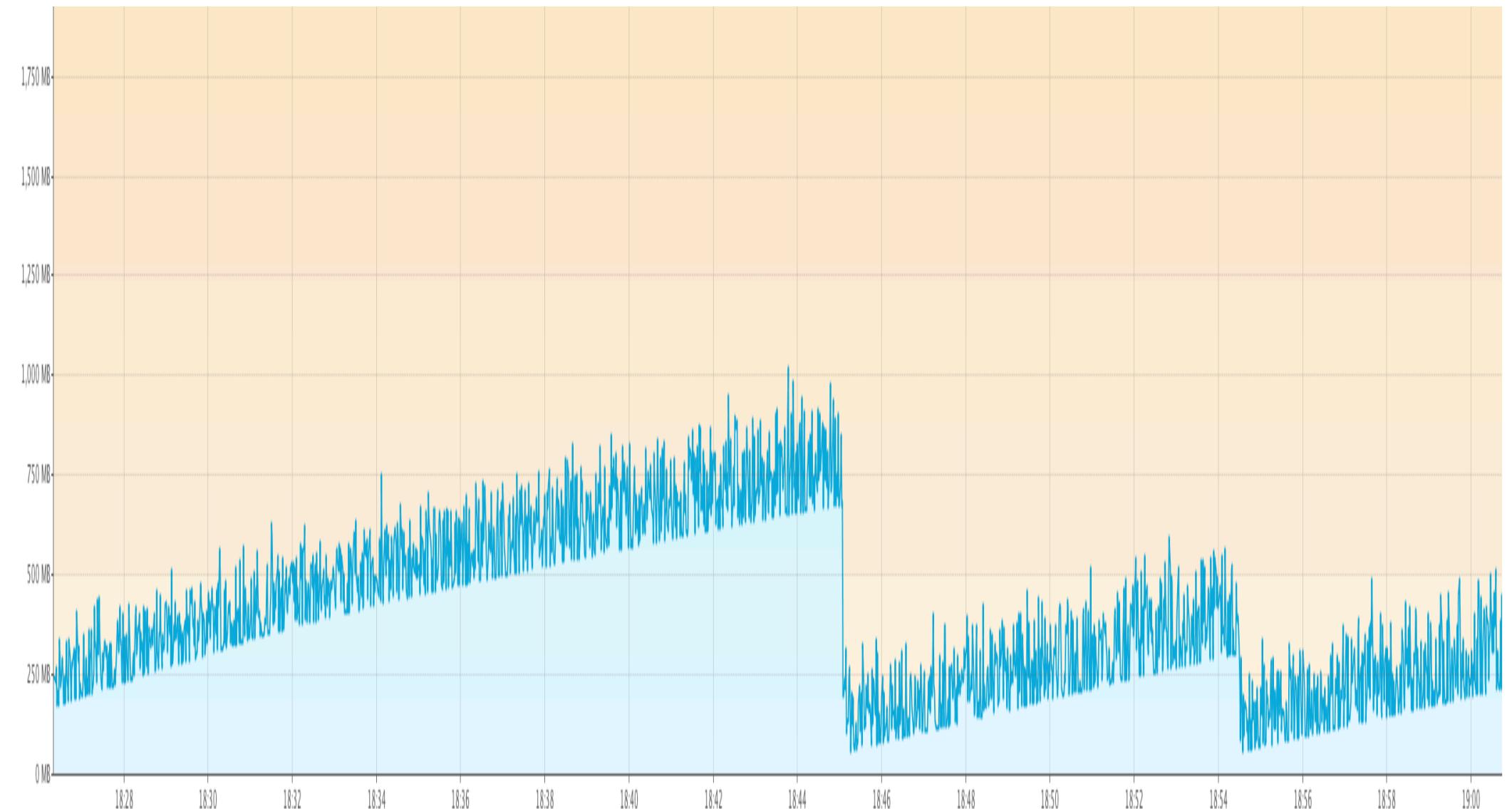
Number of generated processes in the log: 75211

# Memory Profile - QEA



Maximal used heap: 550MB

# Memory Profile - CSP\_E



Maximal used memory: 1G

# Syntax of CSP\_E in BNF

$P ::= \text{SKIP} \mid \text{STOP} \mid \text{Failure} \mid$

$e \rightarrow: P \mid$

$\text{?? } f \mid \text{??? } f \mid$

$P \text{ <+> } P \mid$

$P \parallel a \parallel P \mid$

$P \parallel\parallel P \mid$

$P \$ P$

# Formal Semantics

Trace semantics of Hoare's CSP

Minimal Element  $\llbracket \text{STOP} \rrbracket = \{\langle \rangle\} \in \mathcal{T}$

---

Trace semantics of CSP\_E

Minimal Element  $\llbracket \text{Failure} \rrbracket = \emptyset \in \mathcal{T}$

# StraceMatch

- Analyze system calls
- Verify correctness of file descriptor operations
- Complete specification of Unix API related to processes and file descriptors by CSP\_E

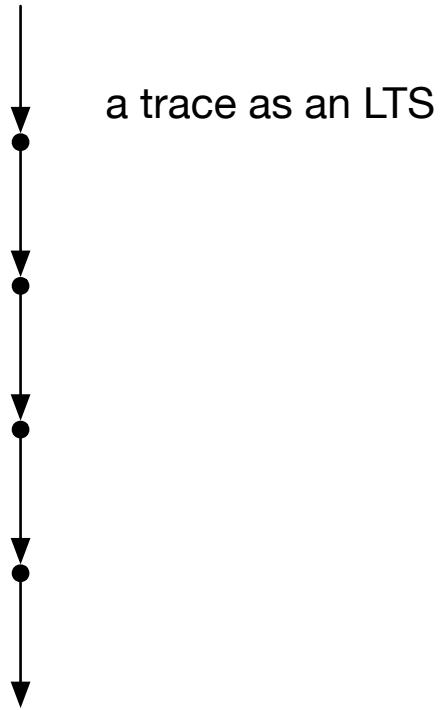
# StraceMatch result

Program	Log size	Result	Time [s]
ls	156	Passed	0.370
wget (short)	267	Fd not closed	0.377
wget (long)	28,901	Fd not closed	1.145
Emacs	2,678	Fd not closed	0.750
Chrome	166,090	Stopped at 2,935	0.810
Ruby ehttpd	11,034	Closed fd twice at 1,168	0.648
Sinatra	3,191	Closed fd twice at 1,170	0.673
bash	1,218	Closed fd twice at 946	0.575

# Monitoring by a model checker

Specification  $\parallel X$

$X$  : observable behaviors



## Difference

- Heavy weight vs. light weight
- Standalone vs. embedded DSL
- CADP's Exhibitor and Evaluator are based on regex. and  $\mu$ -calculus

# LOLA: Runtime Monitoring of Synchronous Systems

$$s_1 = e_1(t_1, \dots, t_m, s_1, \dots, s_n)$$

...

$$s_n = e_n(t_1, \dots, t_m, s_1, \dots, s_n)$$

$s_i$  : output stream

$t_i$  : input stream

## Difference

- Synchronous vs. Asynchronous
- Declarative vs. Operational

# (Poly)LARVA

- Automata-based approach
- Uses ensemble of communicating automaton
- Real-time capability
- Distributed monitoring

## Difference

- Automata-based vs. Process calculus-based
- Static number of automata vs. Dynamic generation of monitors

# Future Works

- Error handling and recovering
- Performance improvement for a large system
- Support for a distributed system (network delay, connection loss, no centralized clock...)
- Distributed monitoring

# Conclusion

We propose

- a monitoring technique for concurrent, asynchronous system
- based on CSP with a minimum change

We showed

- The approach is technically feasible