PARALLEL GESTURE RECOGNITION WITH SOFT REAL-TIME GUARANTEES

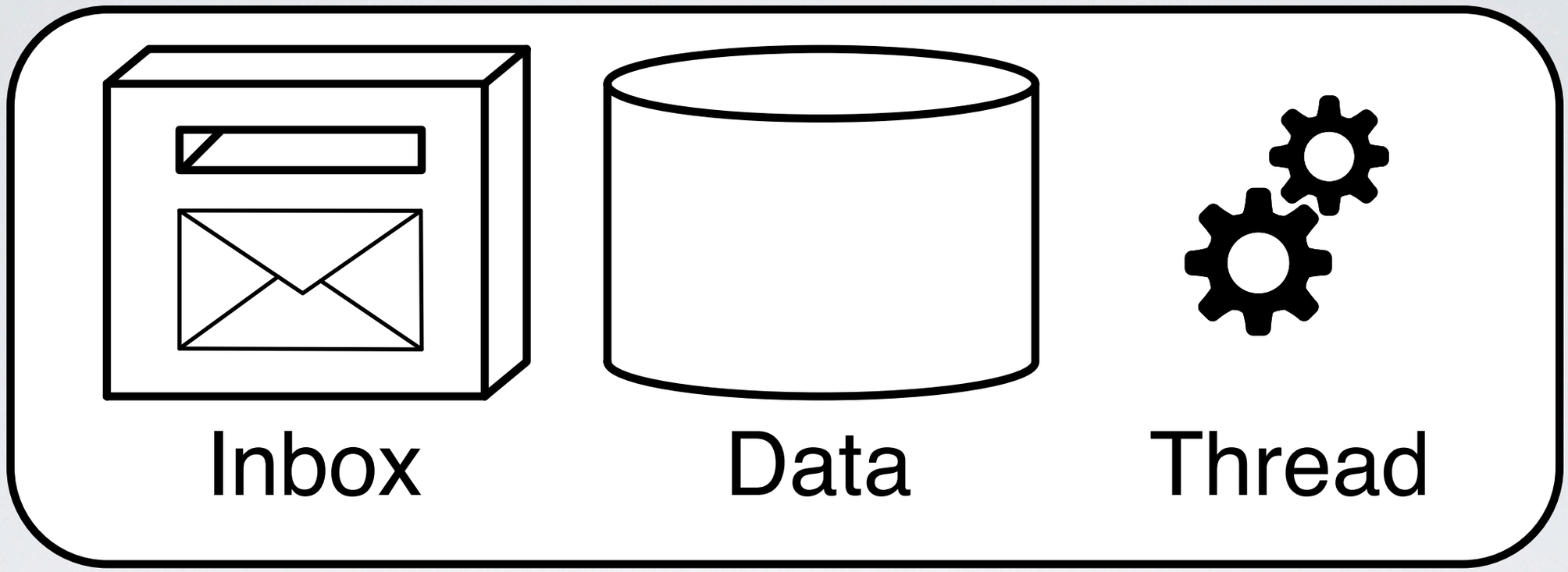
Thierry Renaux
Lode Hoste
Stefan Marr
Wolfgang De Meuter

Software Languages Lab

Vrije Universiteit Brussel

Monday October 22nd, 2012 – AGERE! 2012

SITUATING WITHIN THE ACTOR CONTEXT

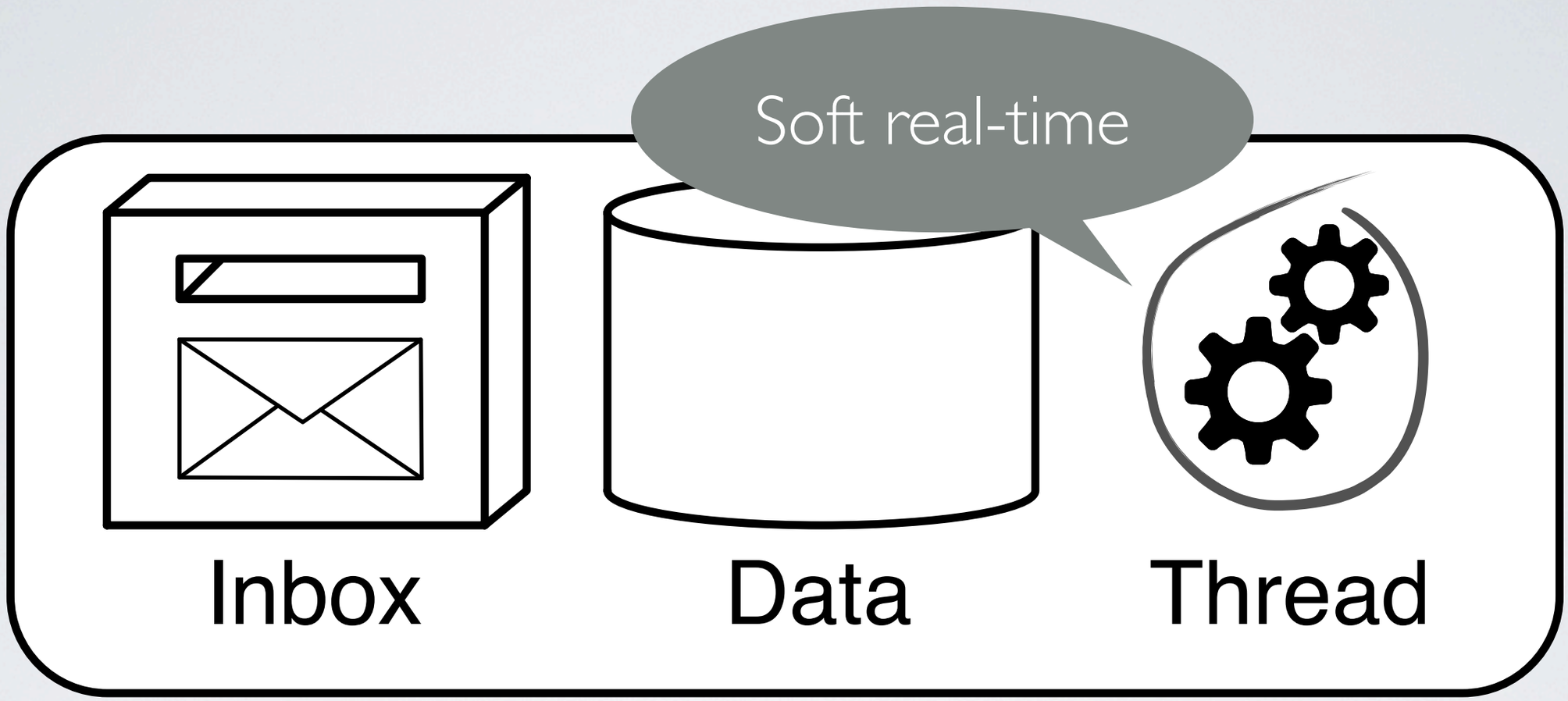


Asynchronous, message passing interoperation

Private state

Own thread of control

SITUATING WITHIN THE ACTOR CONTEXT

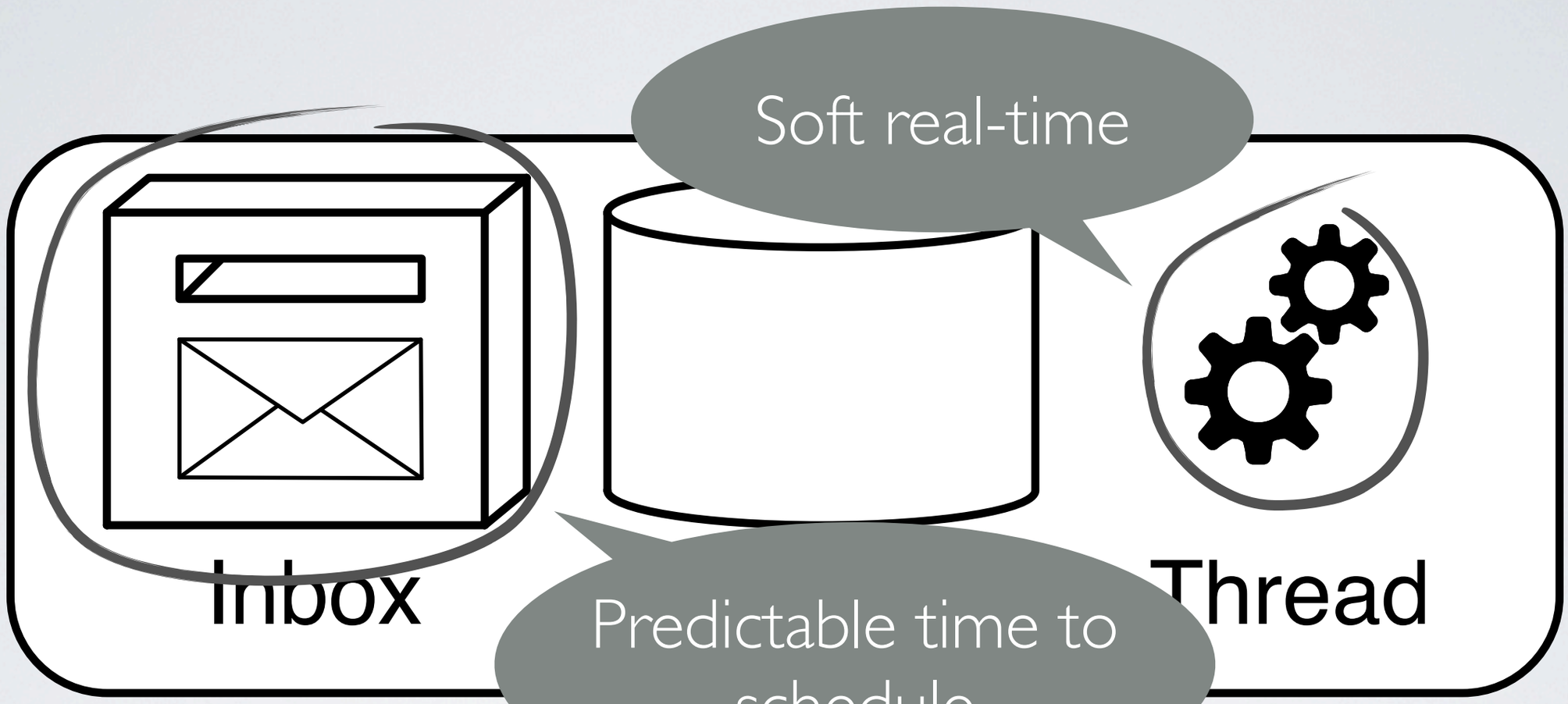


Asynchronous, message passing interoperation

Private state

Own thread of control

SITUATING WITHIN THE ACTOR CONTEXT



Asynchronous, message passing interoperation

Private state

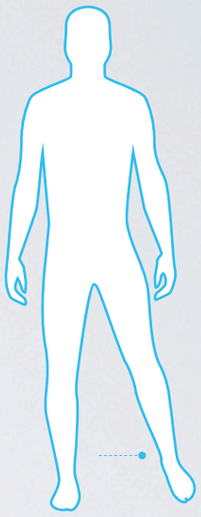
Own thread of control

USE CASE: GESTURE RECOGNITION

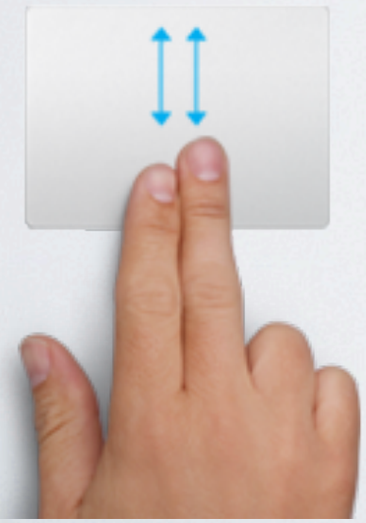
What we get:

Coordinates of touches and releases

Positions of joints in time



What we want:



"A two-finger drag at time t"

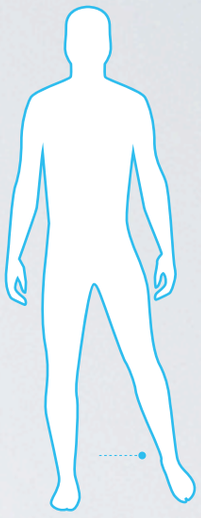
"User u lifted his right leg, with his hands in his sides"

USE CASE: GESTURE RECOGNITION

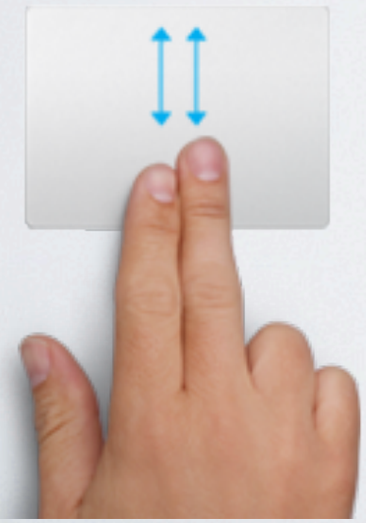
What we get:

Coordinates of touches and releases

Positions of joints in time



What we want:



"A two-finger drag at time t"

"User u lifted his right leg, with his hands in his sides"

USE CASE: APPROACH

```
/*************************************************************************
 *
 * grail - Gesture Recognition And Instantiation Library
 *
 * Copyright (C) 2010 Canonical Ltd.
 * Copyright (C) 2010 Henrik Rydberg <rydberg@bitmath.org>
 *
 * This program is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or (at your
 * option) any later version.
 *
 * This program is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 *************************************************************************/

#include "grail-recognizer.h"
#include <math.h>
#include <stdio.h>

static const int getype[DIM_TOUCH + 1] = {
                0,
                0,
                GRAIL_TYPE_PINCH2,
                GRAIL_TYPE_PINCH3,
                GRAIL_TYPE_PINCH4,
                GRAIL_TYPE_PINCH5,
};

static const int fm_mask = 0x04;

static void set_props(const struct gesture_inserter *gin,
                      struct combo_model *s,
                      const struct move_model *m,
                      const struct utouch_frame *frame)
{
                s->prop[GRAIL_PROP_PINCH_DR] = gin->scale_r * m->fm[FM_R].action_delta;
                s->prop[GRAIL_PROP_PINCH_VR] = gin->scale_r * m->fm[FM_R].velocity;
                s->prop[GRAIL_PROP_PINCH_R] = gin->scale_r * m->fm[FM_R].value;
                s->nprop = 3;
                s->nprop += gin_add_contact_props(gin, s->prop + s->nprop, frame);
}

int gru_pinch(struct grail *ge,
              const struct utouch_frame *frame)
{
                struct gesture_recognizer *gru = ge->gru;
                struct combo_model *state = &gru->pinch;
                struct move_model *move = &gru->move;
                int mask = state->active ? (move->active & fm_mask) : fm_mask;
                if (!move->multi && !move->single) {
                                if (state->active) {
                                                gru_end(ge, state->gid, move,
                    state->prop, state->nprop);
                                                state->active = 0;
                                }
                                return 0;
                }
                if ((move->timeout & fm_mask) == fm_mask) {
                                if (state->active) {
                                                gin_gid_discard(ge, state->gid);
                                                state->active = 0;
                                }
                                return 0;
                }
                if (!(move->tickle & mask))
                                return 0;
                if (!state->active) {
                                int type = getype[move->ntouch];
                                if (!type)
                                                return 0;
                                state->gid = gin_gid_begin(ge, type, PRIO_GESTURE, frame);
                                state->active = 1;
                }
                if (!(move->active & fm_mask))
                                return 0;
                set_props(ge->gin, state, move, frame);
                gru_event(ge, state->gid, move, state->prop, state->nprop);
                return 1;
}

int gru_winpinch(struct grail *ge,
                 const struct utouch_frame *frame)
{
                struct gesture_recognizer *gru = ge->gru;
                struct combo_model *state = &gru->winpinch;
                struct move_model *move = &gru->move;
                int mask = state->active ? (move->active & fm_mask) : fm_mask;
                if (!move->multi) {
                                if (state->active && out_of_bounds(state, move)) {
                                                gru_end(ge, state->gid, move,
                    state->prop, state->nprop);
                                                state->active = 0;
                                }
                                return 0;
                }
                if ((move->timeout & fm_mask) == fm_mask) {
                                if (state->active) {
                                                gin_gid_discard(ge, state->gid);
                                                state->active = 0;
                                }
                                return 0;
                }
                if (!(move->tickle & mask))
                                return 0;
                if (!state->active) {
                                if (move->ntouch == 4) {
                                                state->gid = gin_gid_begin(ge,
GRAIL_TYPE_MPINCH,
                                                PRIO_META, frame);
                                                state->mintouch = 2;
                                                state->maxtouch = 4;
                                                state->active = 1;
                                } else if (move->ntouch == 3) {
                                                state->gid = gin_gid_begin(ge,
GRAIL_TYPE_EPINCH,
                                                PRIO_ENV, frame);
                                                state->mintouch = 2;
                                                state->maxtouch = 3;
                                                state->active = 1;
                                } else {
                                                return 0;
                                }
                }
                if (!(move->active & fm_mask))
                                return 0;
                set_props(ge->gin, state, move, frame);
                gru_event(ge, state->gid, move, state->prop, state->nprop);
                return 1;
}
```

Approaches:

- Imperative
  - Cumbersome
  - Error-prone
- Machine Learning
  - Learning phase
  - Debuggability?!
- Declarative

USE CASE: APPROACH

```
(defrule pinchGesture
 (FingerMoved (startPoint ?sIndex)
              (endPoint ?eIndex)
              (timestamp ?tIndex)
              (duration ?deltaTIndex)
              (finger "index"))
 (FingerMoved (startPoint ?sThumb)
              (endPoint ?eThumb)
              (timestamp ?tThumb)
              (duration ?deltaTThumb)
              (finger "thumb"))
 (test (< (euclideanDistance ?eIndex ?eThumb)
          (euclideanDistance ?sIndex ?sThumb)))
 (test (overlapping ?tIndex ?tThumb
                    ?durIndex ?durThumb))
=>
 (reactToPinchAt ?sIndex ?tIndex)))
```

Approaches:

- Imperative
  - Cumbersome
  - Error-prone
- Machine Learning
  - Learning phase
  - Debuggability?!
- Declarative

USE CASE: THE RETE ALGORITHM

Declarative approach

Algorithms such as Rete,
Lana, Leaps, GenTrie, ...

```
; Detect Tap gesture
(defrule detectTap
     (Touch (location ?startLoc) (timestamp ?t1))
     (Release (locaction ?startLoc) (timestamp ?t2))
     (test (< (abs (- ?t1 ?t2)) MAX_DELAY))
=>
     (reactToTapAt ?startLoc))

; Detect left swipe gesture
(defrule detectLeftSwipe
     (Touch (location ?startLoc) (timestamp ?t1))
     (Release (locaction ?endLoc) (timestamp ?t2))
     (test (< (abs (- ?t1 ?t2)) MAX_DELAY))
     (test (toTheLeftOf ?endLoc ?startLoc))
=>
     (reactToSwipe))
```

USE CASE: THE RETE ALGORITHM

Declarative approach

Algorithms such as Rete,
~~Lana, Leaps, GenTrie~~, ...

- Among fastest sequential algorithms
- Transforms rules into DAG
- Caches partial matches

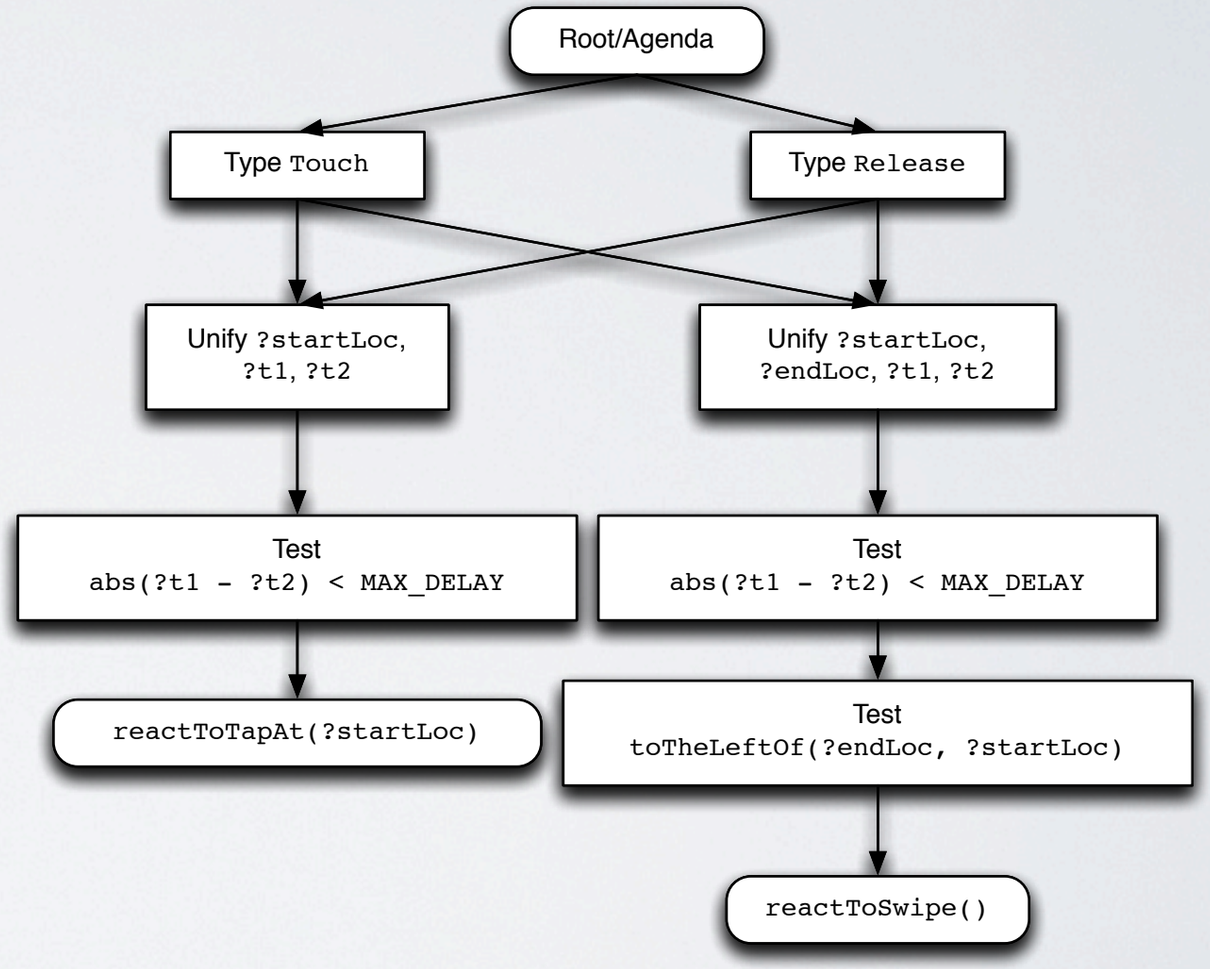
```
; Detect Tap gesture
(defrule detectTap
     (Touch (location ?startLoc) (timestamp ?t1))
     (Release (locaction ?startLoc) (timestamp ?t2))
     (test (< (abs (- ?t1 ?t2)) MAX_DELAY))
=>
     (reactToTapAt ?startLoc))

; Detect left swipe gesture
(defrule detectLeftSwipe
     (Touch (location ?startLoc) (timestamp ?t1))
     (Release (locaction ?endLoc) (timestamp ?t2))
     (test (< (abs (- ?t1 ?t2)) MAX_DELAY))
     (test (toTheLeftOf ?endLoc ?startLoc))
=>
     (reactToSwipe))
```

USE CASE: THE RETE ALGORITHM

Declarative approach

Algorithms such as Rete, ~~Lana, Leaps, GenTrie~~, ...

- Among fastest sequential algorithms
- Transforms rules into DAG
- Caches partial matches

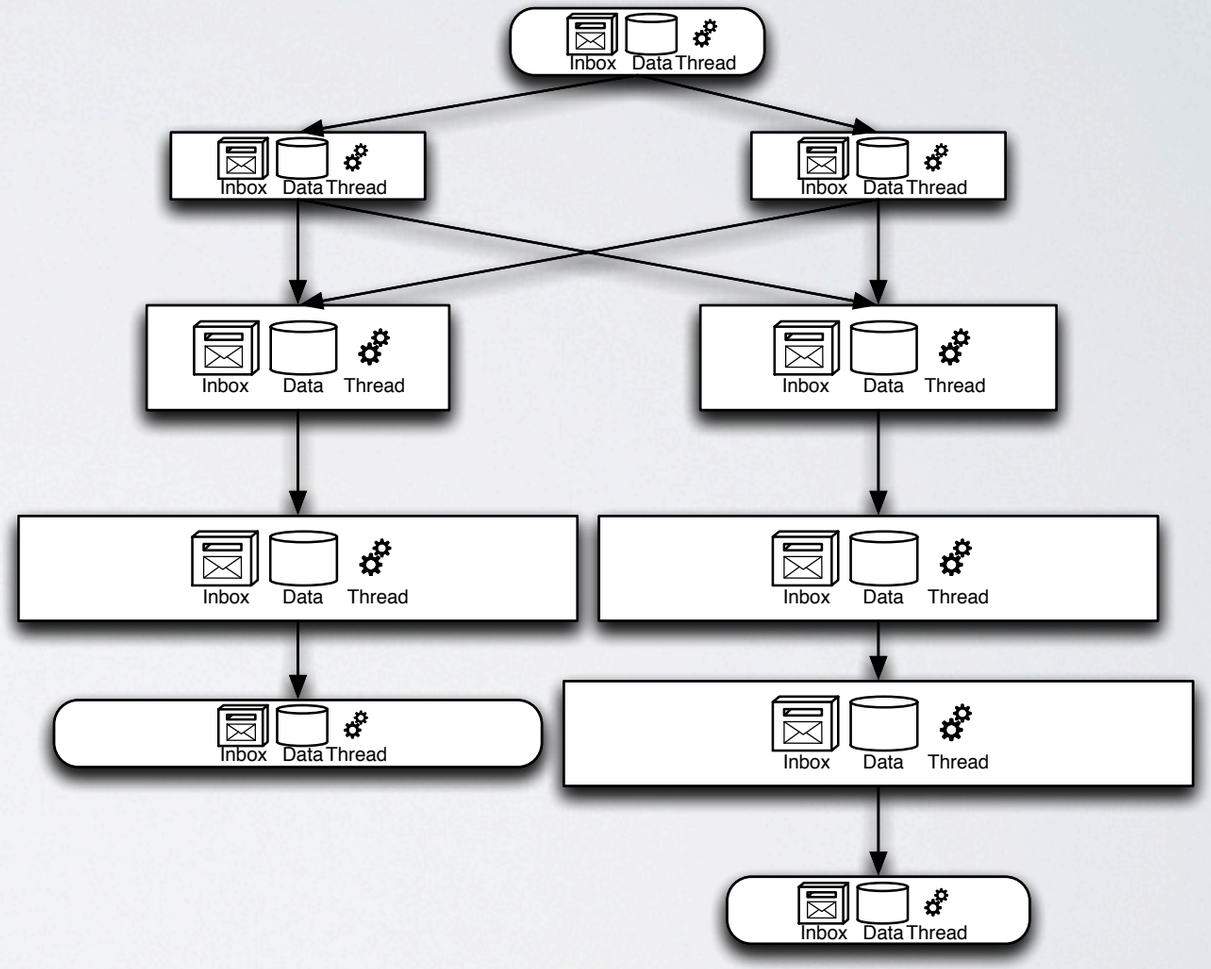


USE CASE: THE RETE ALGORITHM

Declarative approach

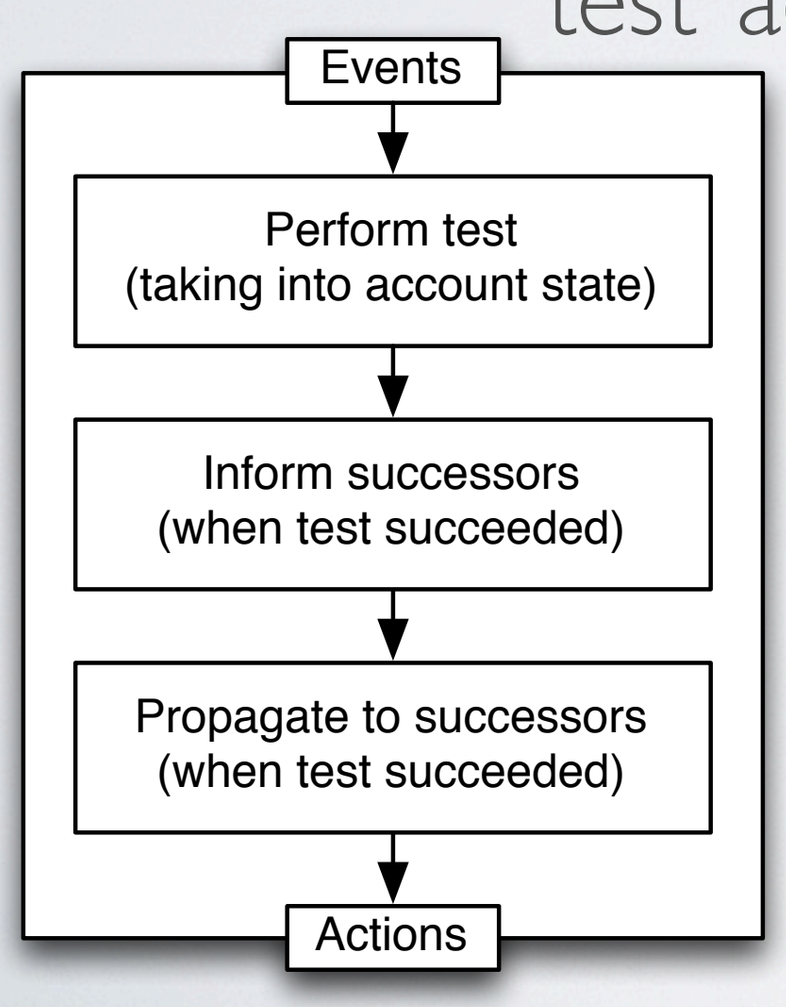
Algorithms such as Rete,
~~Lana, Leaps, GenTrie~~, ...

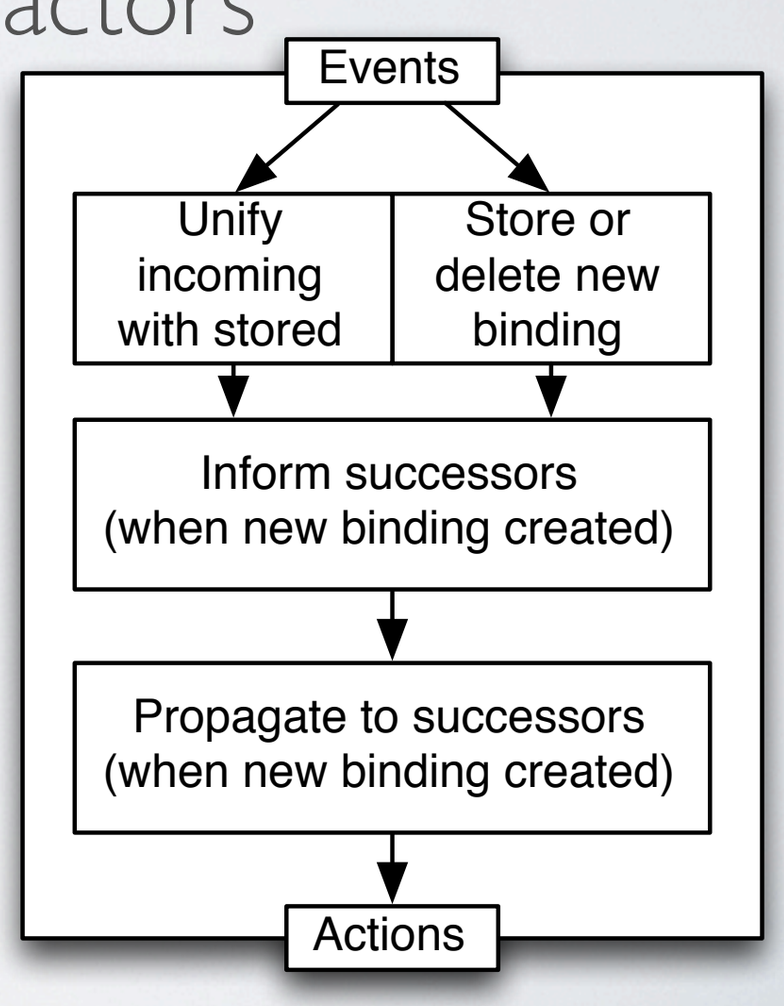
- Among fastest sequential algorithms
- Transforms rules into DAG
- Caches partial matches
- Maps naturally onto actors
  - No shared state
  - Communication explicit
  - But with specific requirements from the application domain



MESSAGES AND ACTIONS

- Only two messages must be understood:
  - **Assert** a new fact
  - **Retract** an old fact





MEMORY MANAGEMENT / EXPIRATION

- Lifetime of facts implicit in the rules
  - If (< (abs (- ?t1 ?t2)) MAX_DELAY)
  - And there has been no candidate for ?t2 for MAX_DELAY
  - Then there's no point in keeping candidates for ?t1

MEMORY MANAGEMENT / EXPIRATION

- Lifetime of facts implicit in the rules
  - If (< (abs (- ?t1 ?t2)) MAX_DELAY)
  - And there has been no candidate for ?t2 for MAX_DELAY
  - Then there's no point in keeping candidates for ?t1

- Keeping track of time at your predecessors
  - Only requires to piggy-back that data in message-sends

MEMORY MANAGEMENT / EXPIRATION

- Lifetime of facts implicit in the rules
  - If (< (abs (- ?t1 ?t2)) MAX_DELAY)
  - And there has been no candidate for ?t2 for MAX_DELAY
  - Then there's no point in keeping candidates for ?t1

- Keeping track of time at your predecessors
  - Only requires to piggy-back that data in message-sends

- Discarding expired facts requires no syncing

MEMORY MANAGEMENT / EXPIRATION

- Lifetime of facts implicit in the rules
  - If (< (abs (- ?t1 ?t2)) MAX_DELAY)
  - And there has been no candidate for ?t2 for MAX_DELAY
  - Then there's no point in keeping candidates for ?t1

- Keeping track of time at your predecessors
  - Only requires to piggy-back that data in message-sends

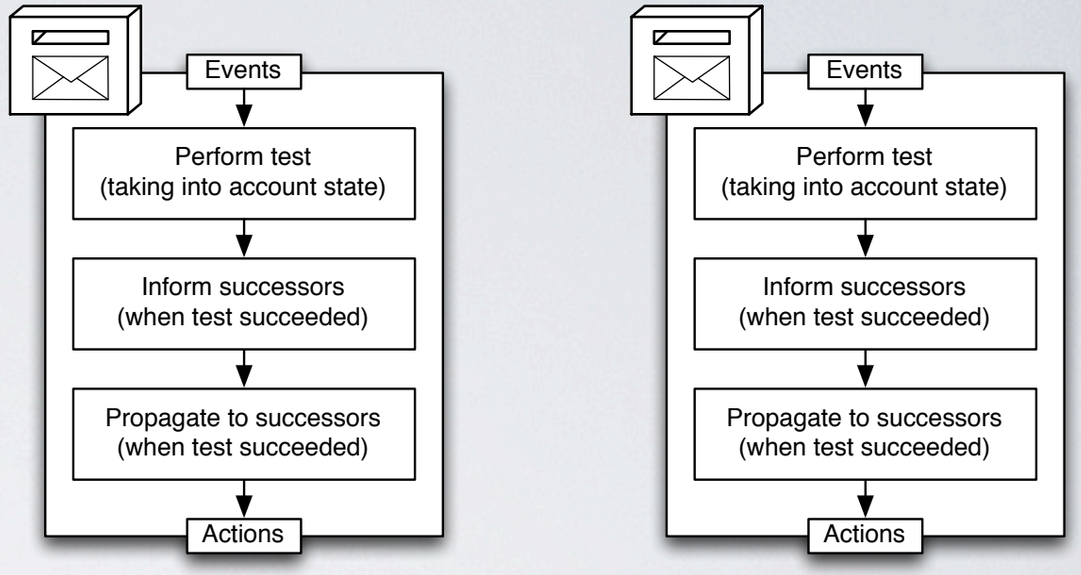
- Discarding expired facts requires no syncing
- But this also means our actors only respond to a single message...

DEADLINES

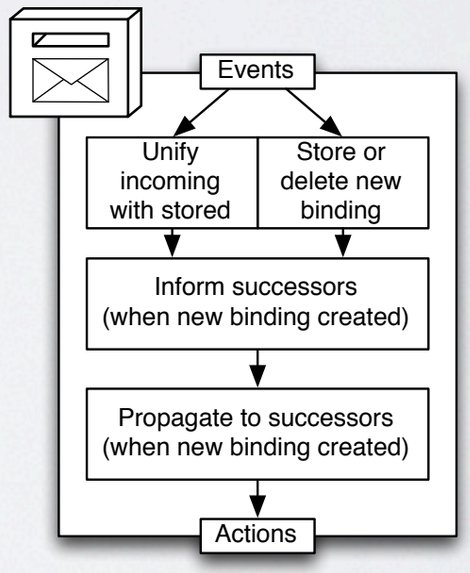
- Guaranteed low latency
- Semantics for missing a deadline
  - Dropping events

DEADLINES

- Guaranteed low latency



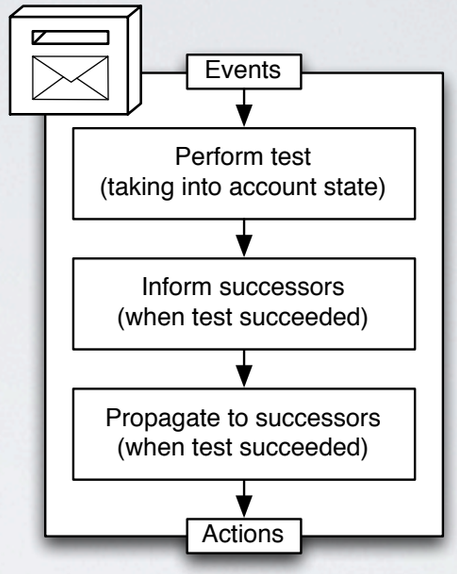
- Semantics for missing a deadline
  - Dropping events

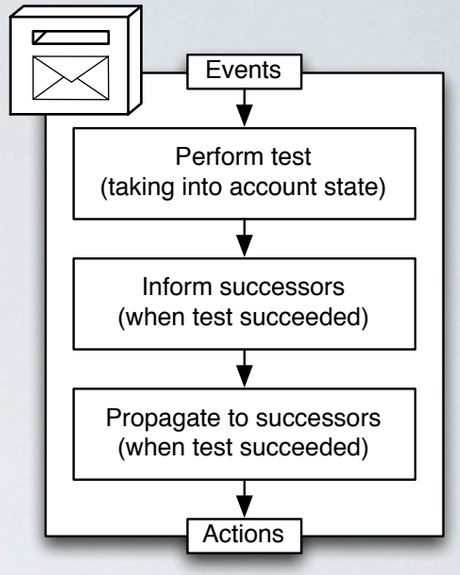


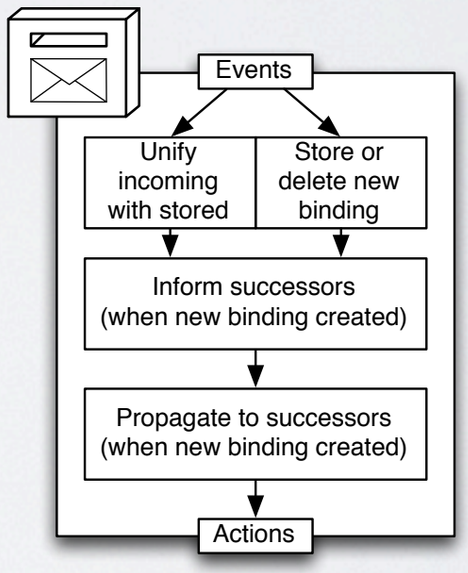
MEETING DEADLINES

Limiting the time spent in

- **Turns** of an actor algorithmically



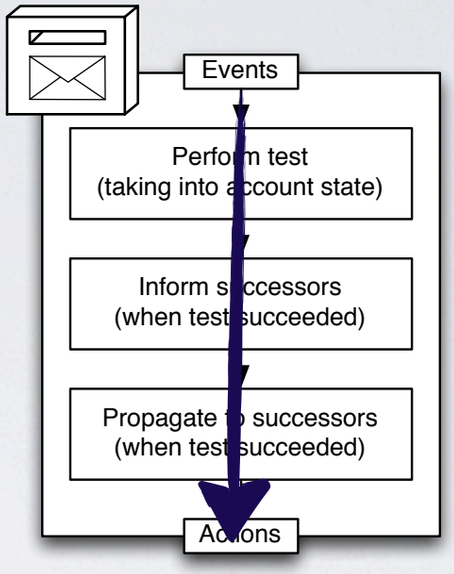


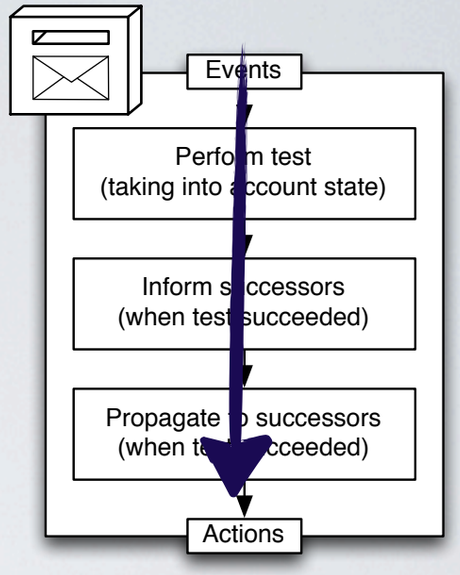


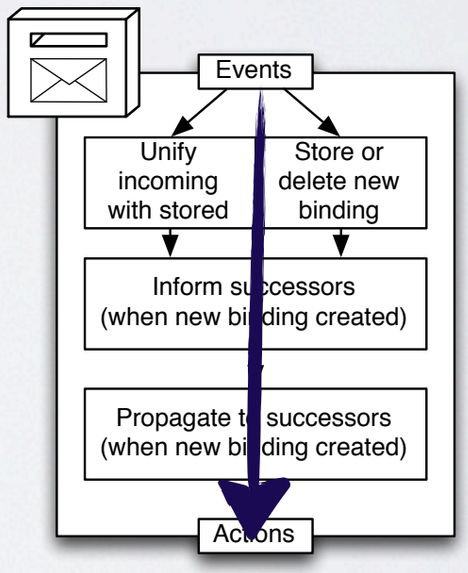
MEETING DEADLINES

Limiting the time spent in

- **Turns** of an actor algorithmically



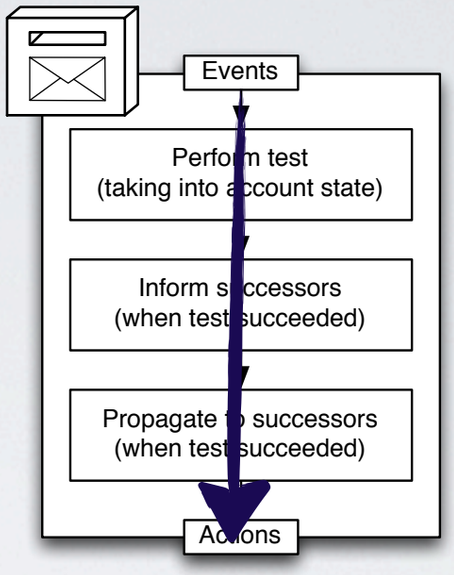


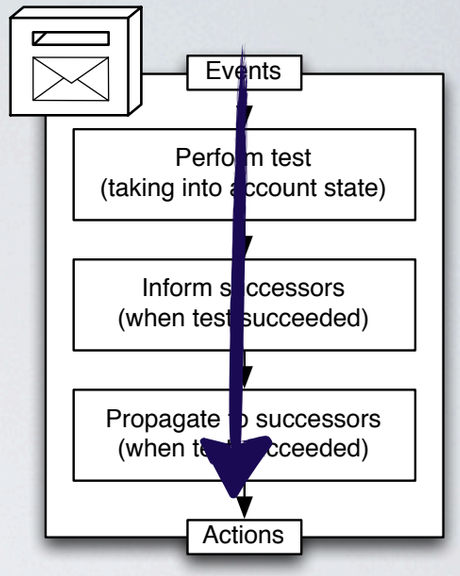


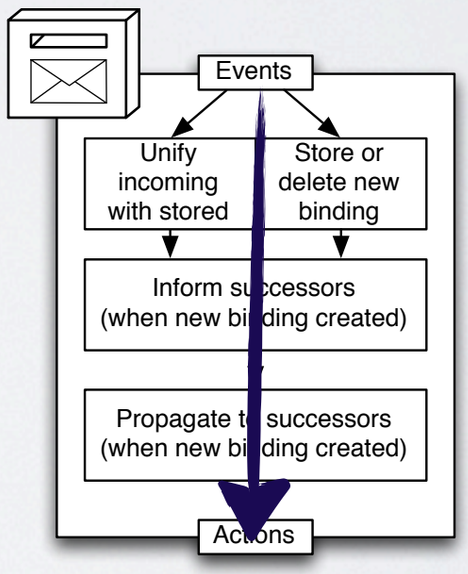
MEETING DEADLINES

Limiting the time spent in

- **Turns** of an actor algorithmically
- **Dequeues** and **enqueues** through use of nonblocking inboxes



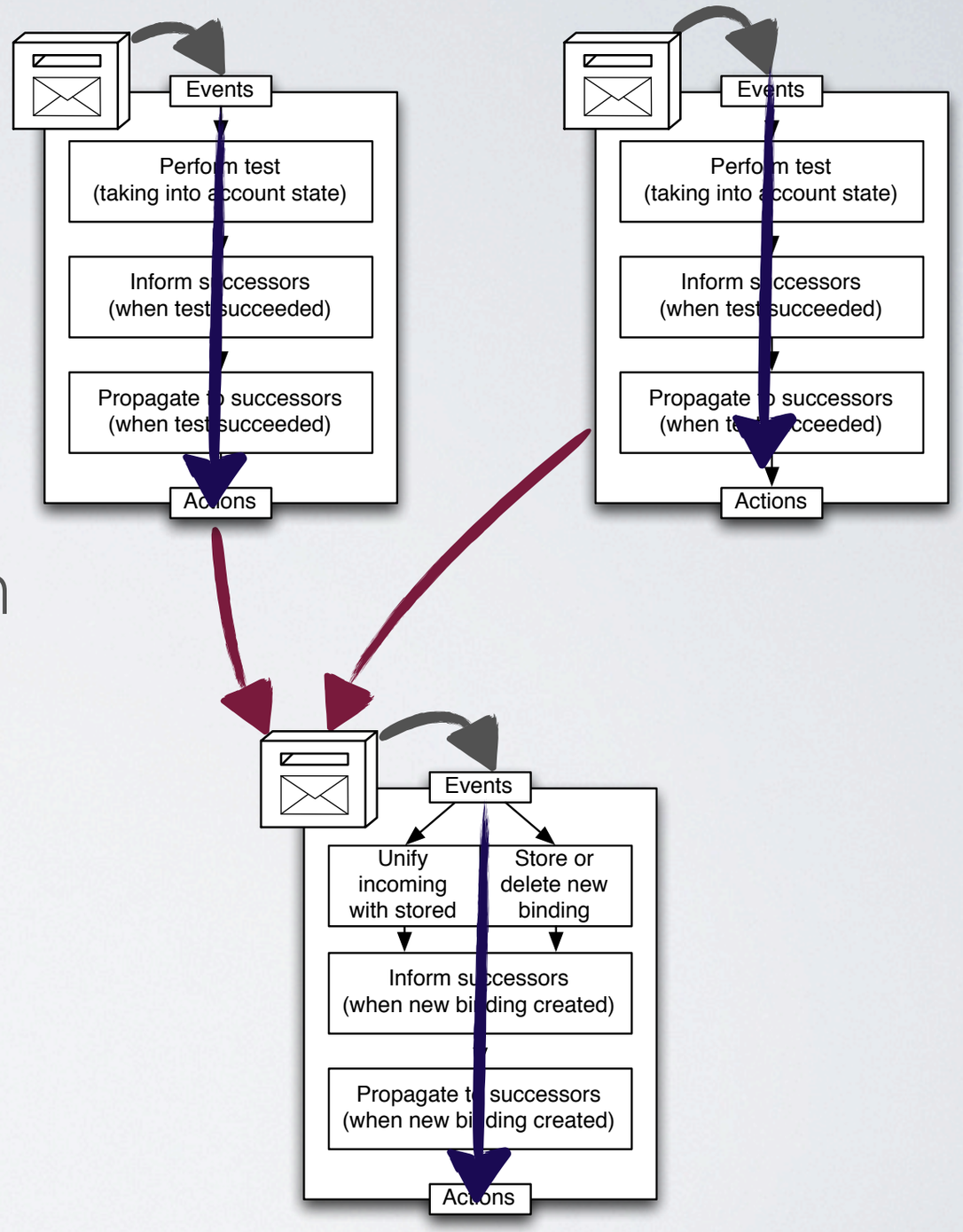




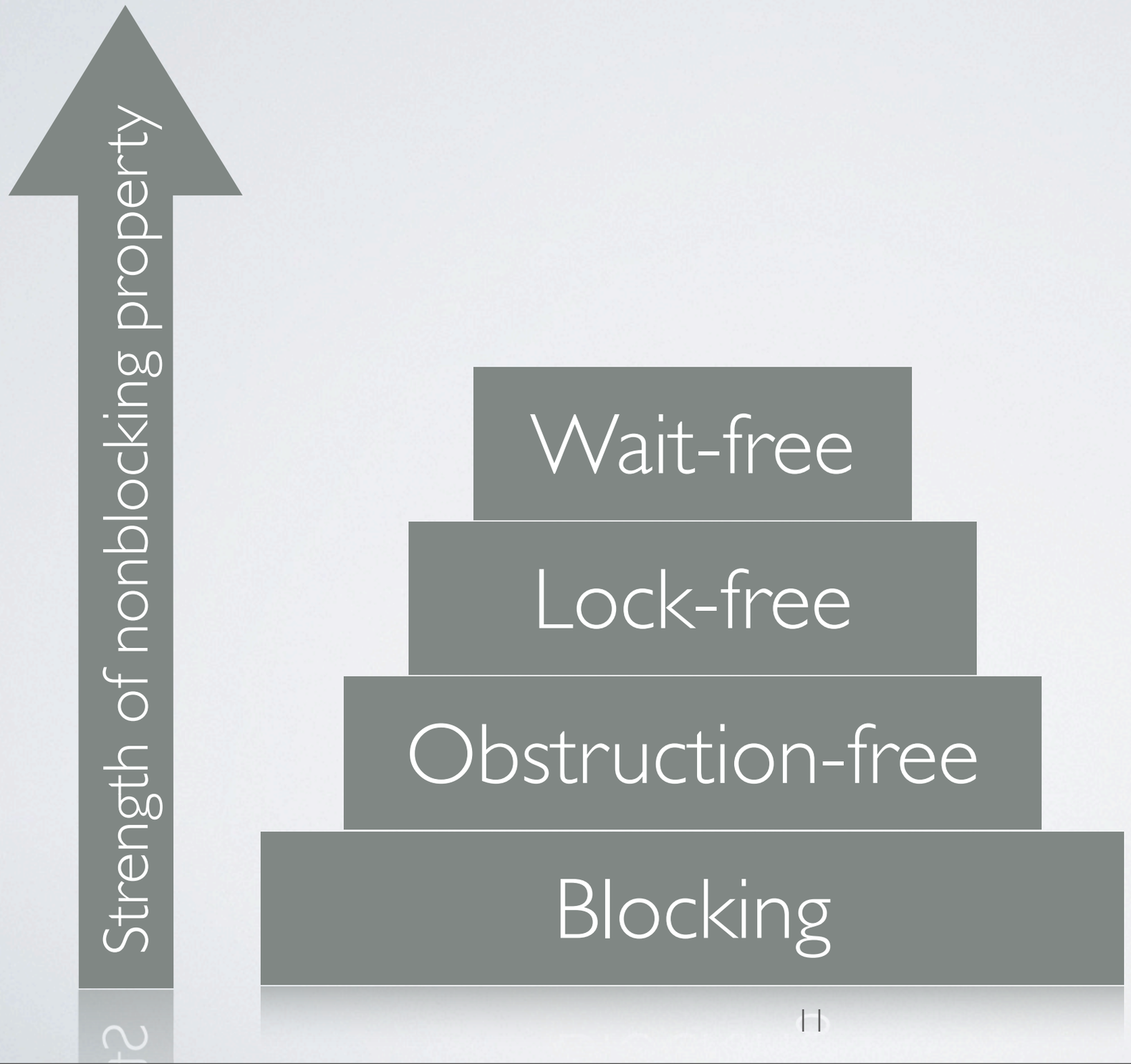
MEETING DEADLINES

Limiting the time spent in

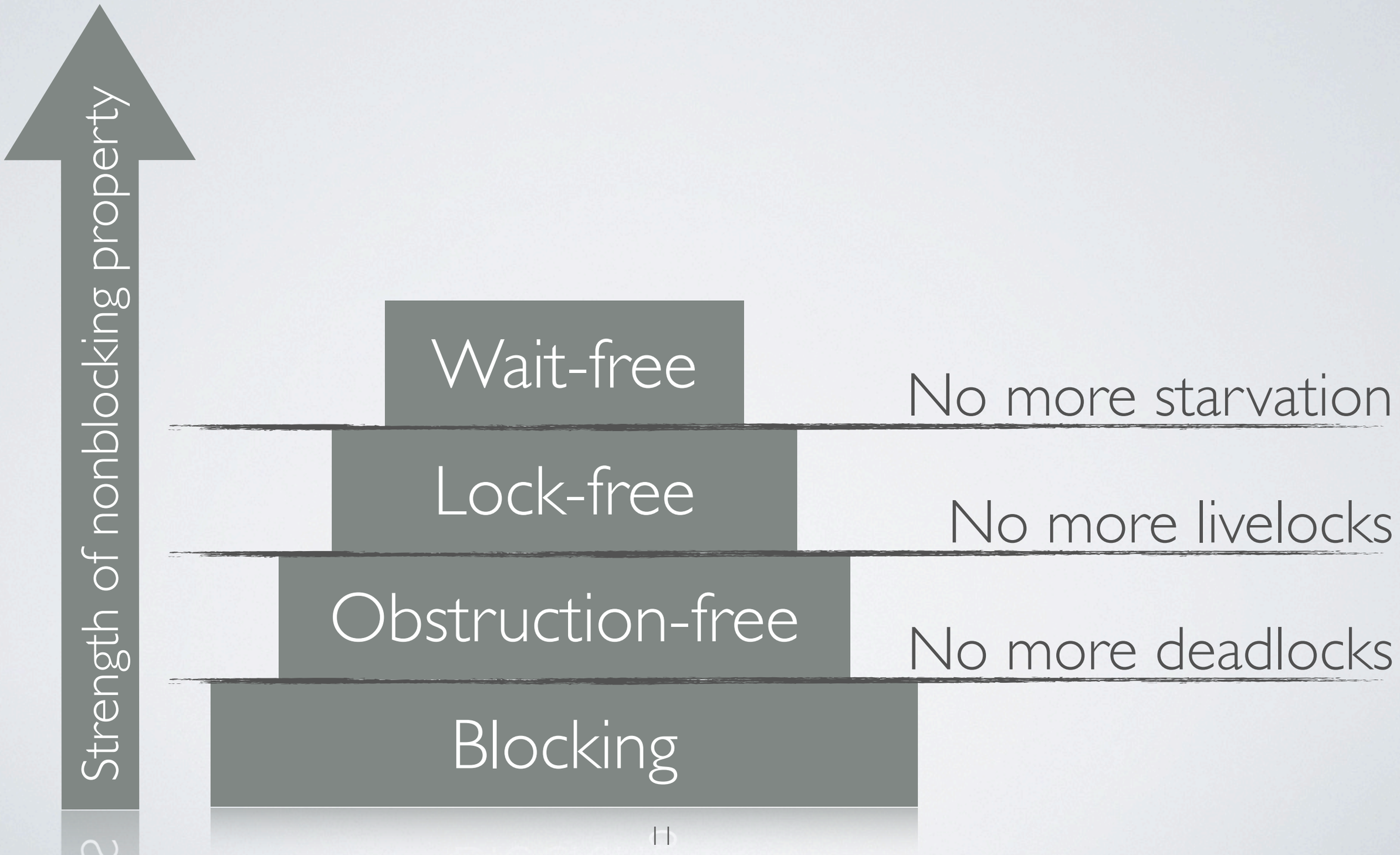
- **Turns** of an actor algorithmically
- **Dequeues** and **enqueues** through use of nonblocking inboxes



NONBLOCKING DATA STRUCTURES



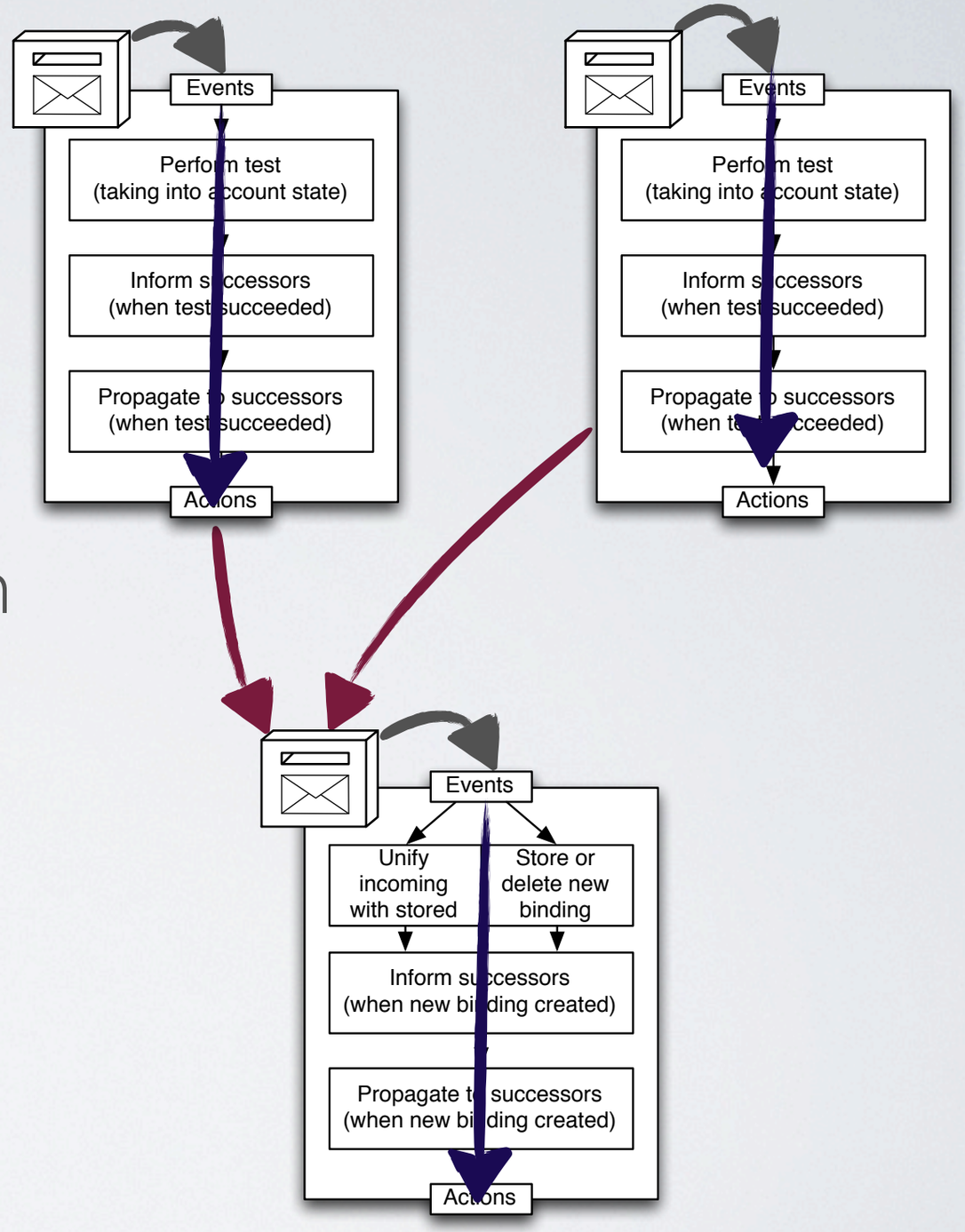
NONBLOCKING DATA STRUCTURES



MEETING DEADLINES

Limiting the time spent in

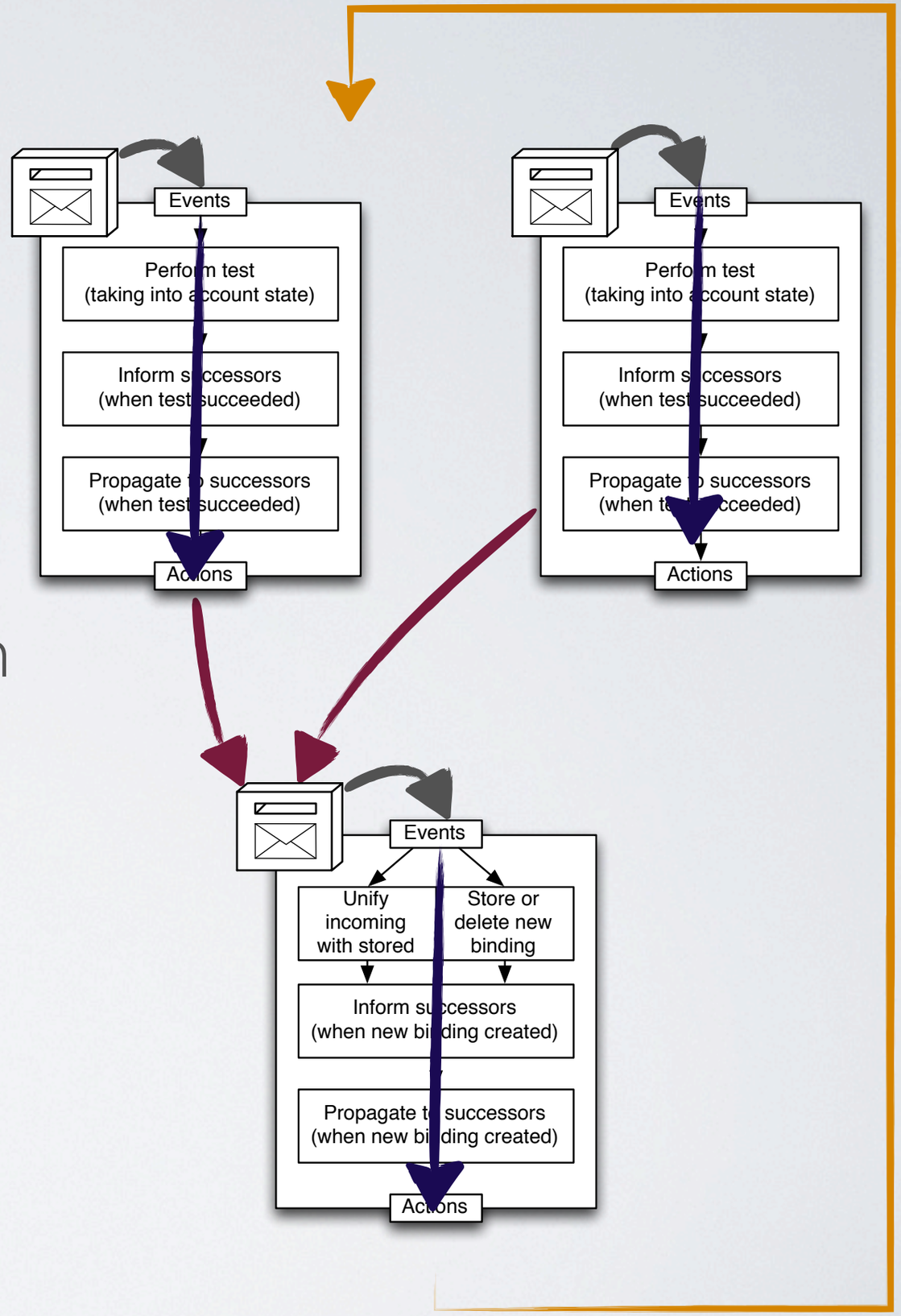
- **Turns** of an actor algorithmically
- **Dequeues** and **enqueues** through use of nonblocking inboxes
- **Loops** over the entire graph by limiting the DSL's expressiveness



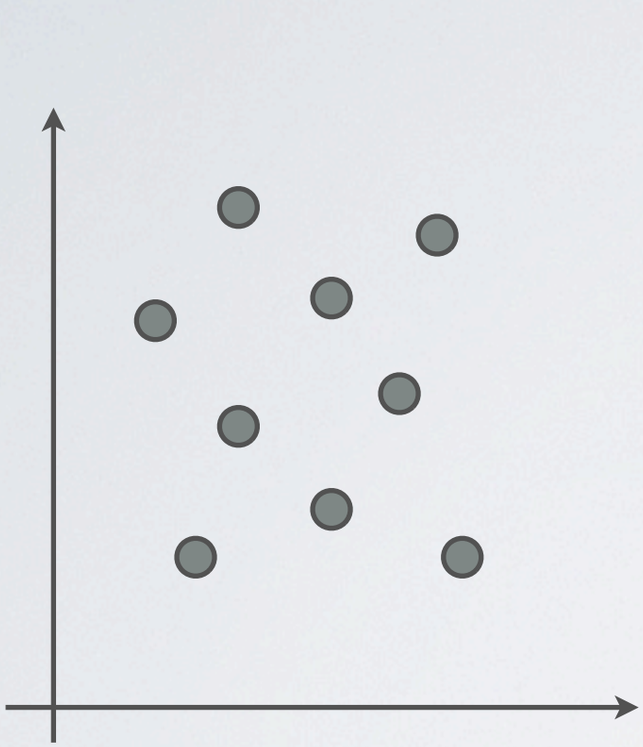
MEETING DEADLINES

Limiting the time spent in

- **Turns** of an actor algorithmically
- **Dequeues** and **enqueues** through use of nonblocking inboxes
- **Loops** over the entire graph by limiting the DSL's expressiveness

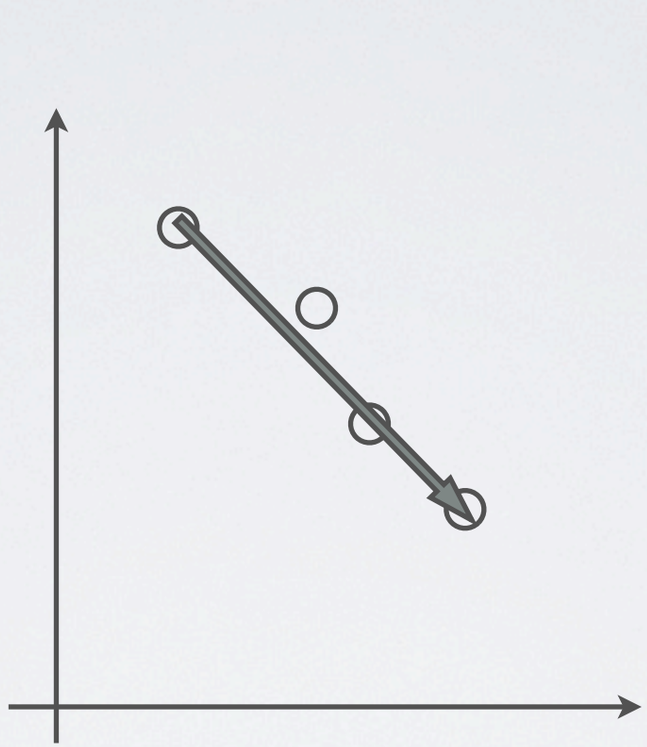


TIERING



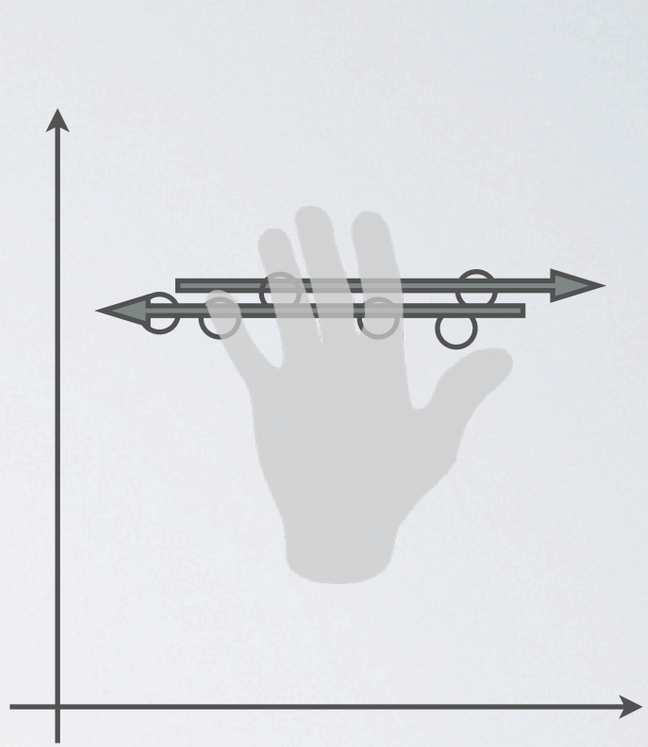
Point

Tier n



Move

Tier n + 1

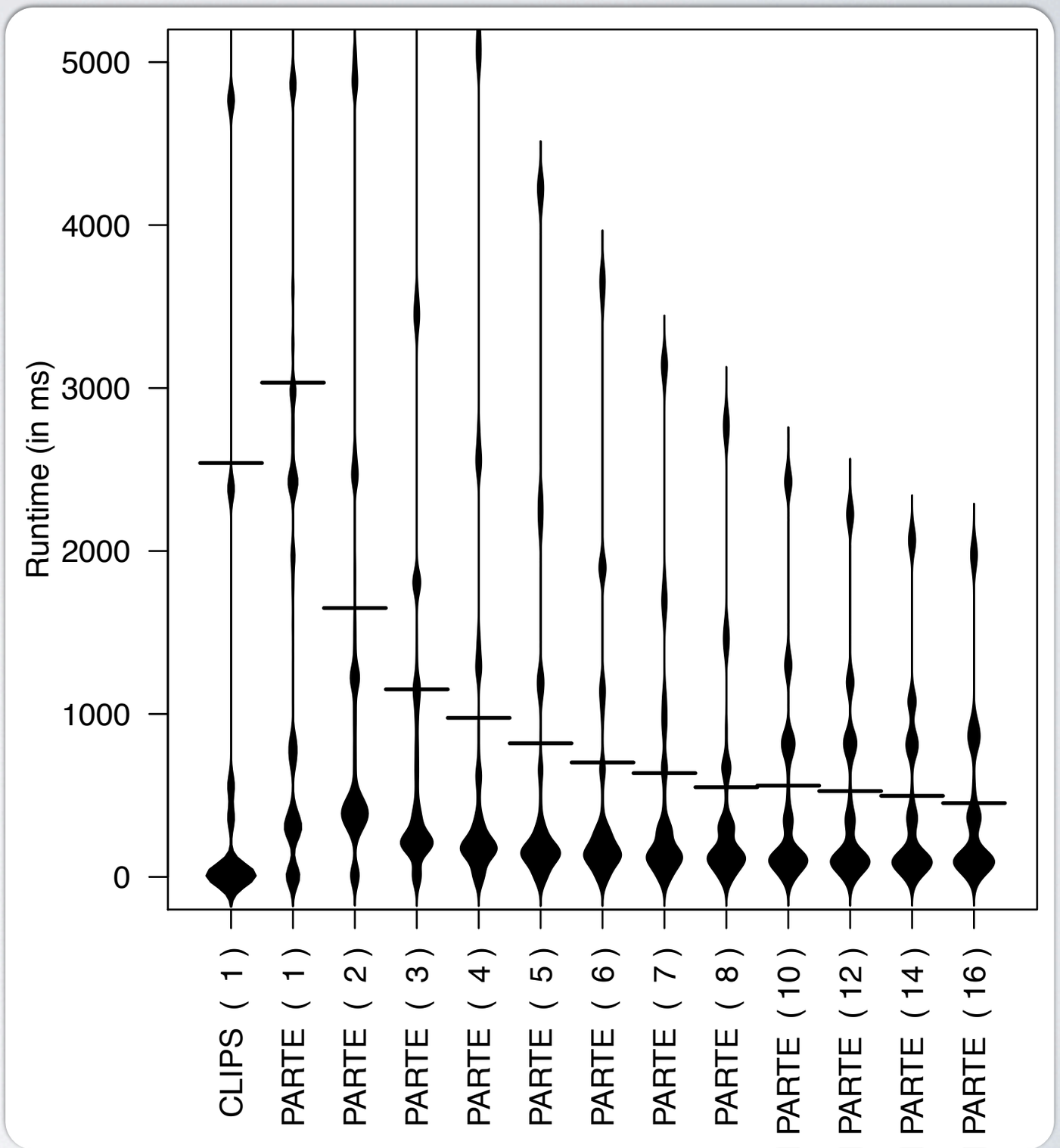


Wave

Tier n + 2

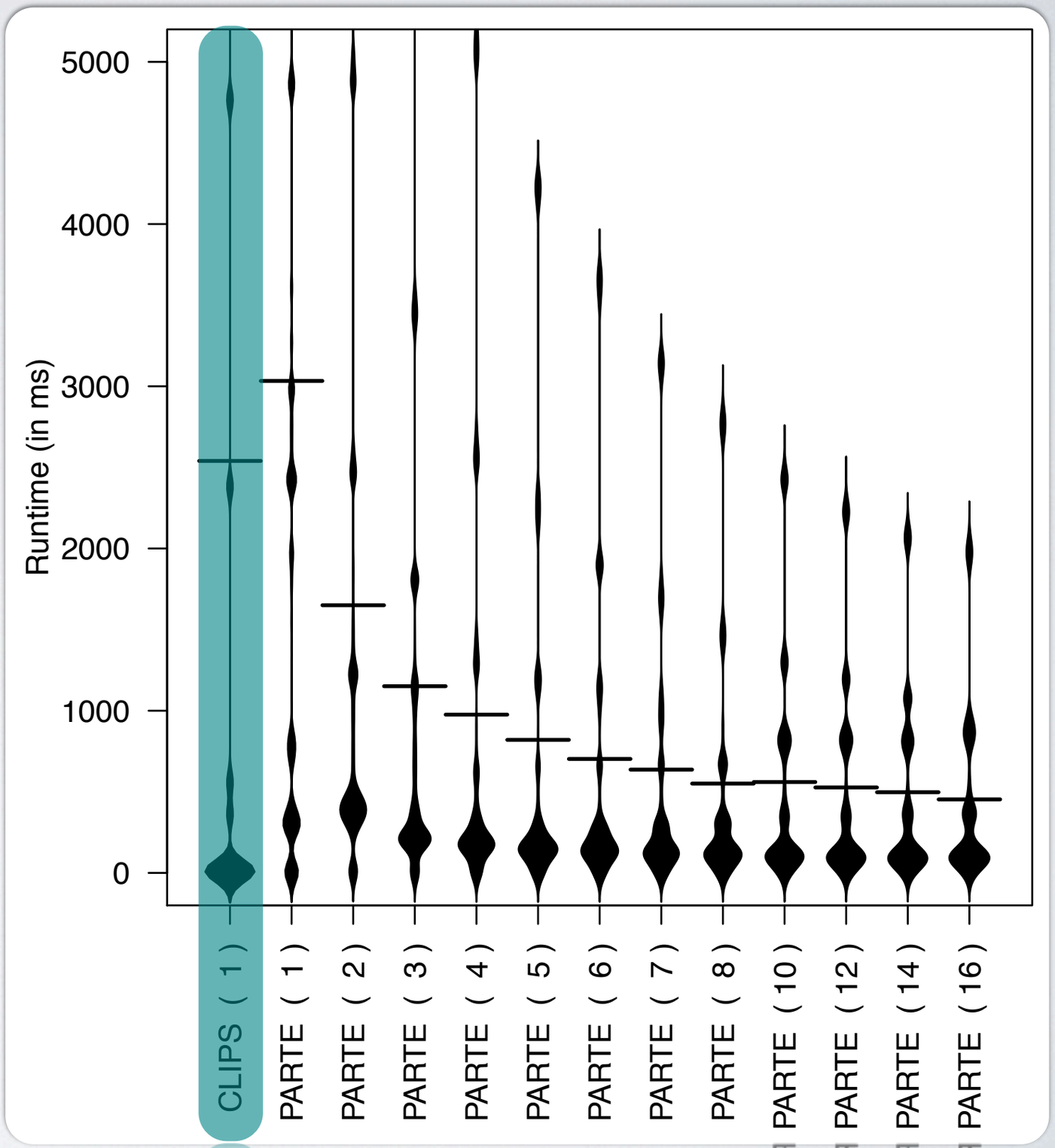
EXECUTION PERFORMANCE

Performance compared to state of the art sequential implementations



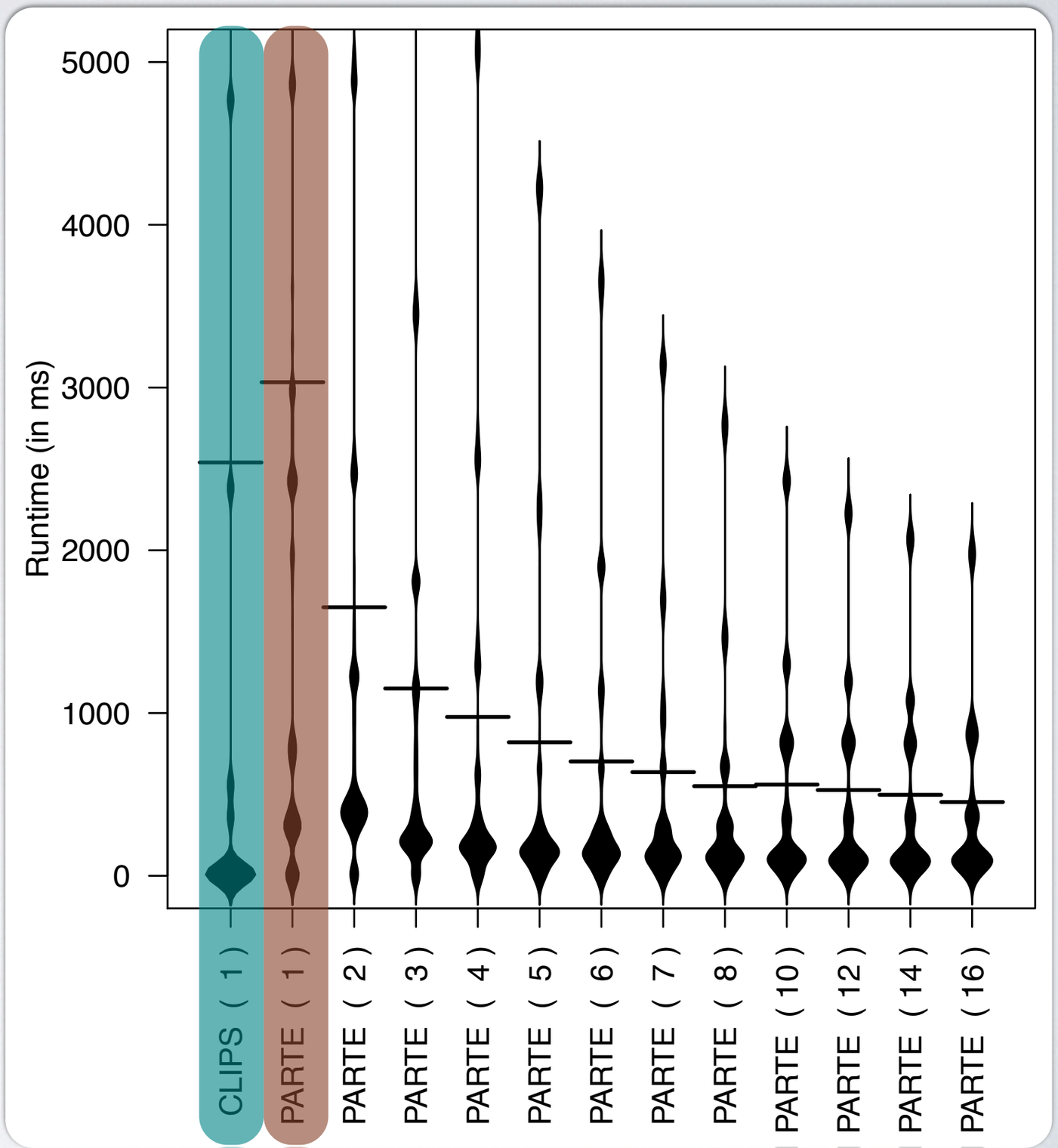
EXECUTION PERFORMANCE

Performance compared to state of the art sequential implementations



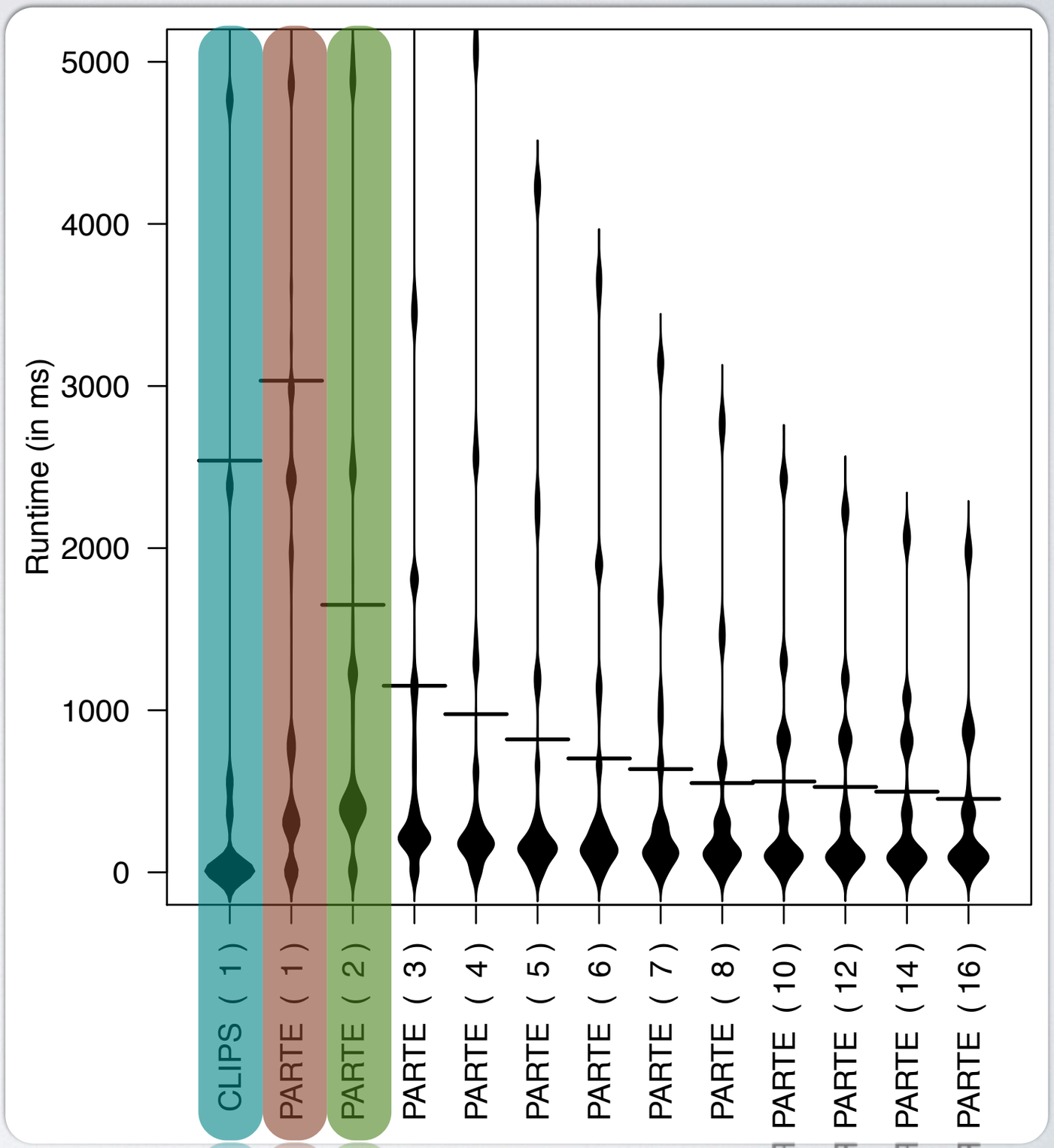
EXECUTION PERFORMANCE

Performance compared to state of the art sequential implementations



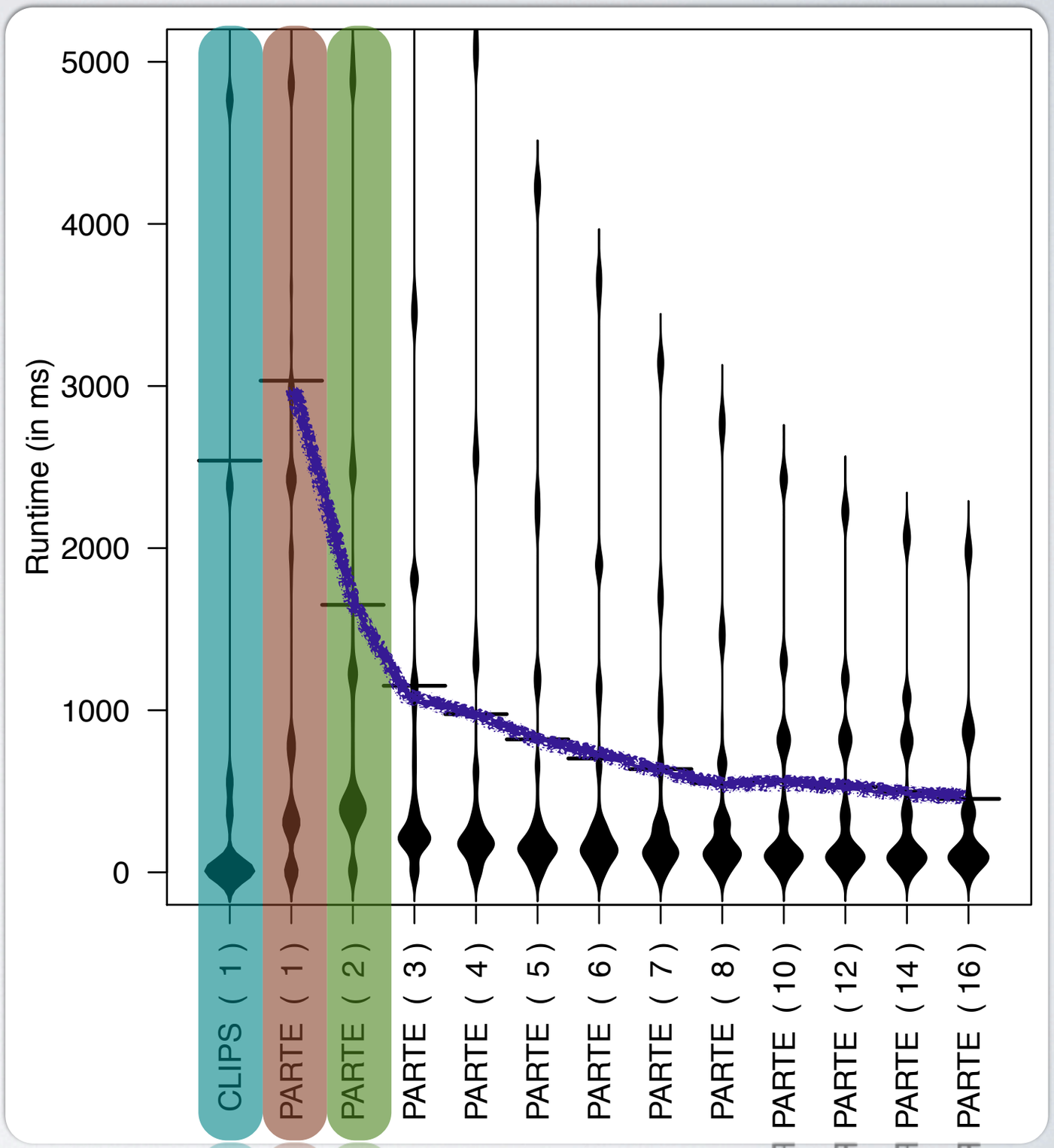
EXECUTION PERFORMANCE

Performance compared to state of the art sequential implementations

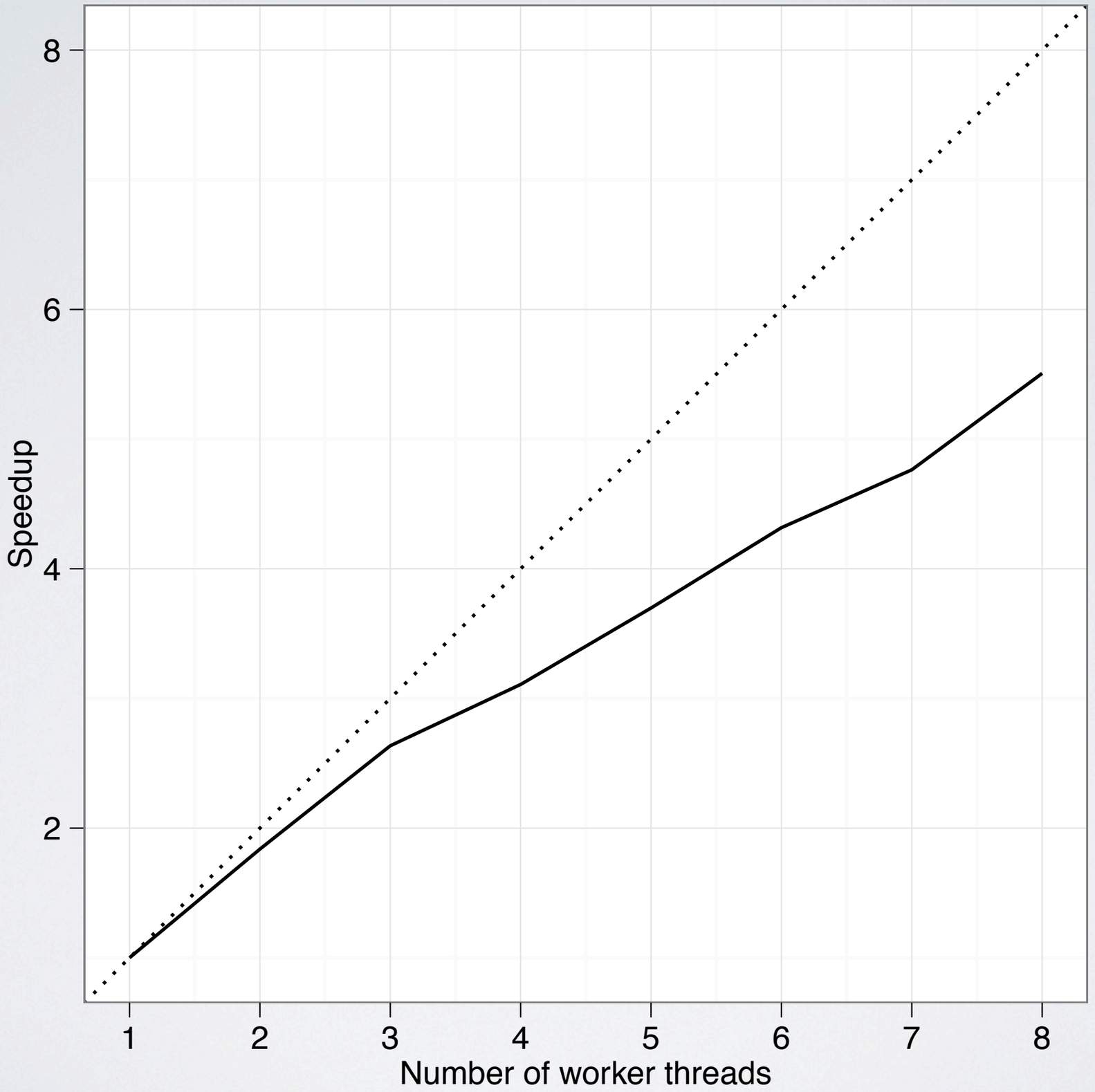


EXECUTION PERFORMANCE

Performance compared to state of the art sequential implementations



SPEEDUP (ON AN 8-CORE MACHINE)

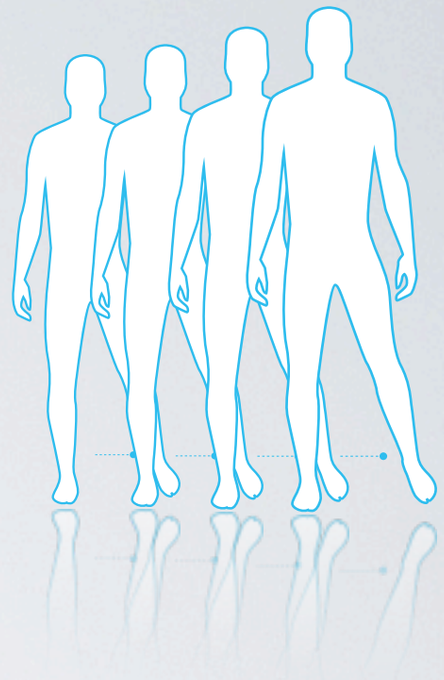


RELATED WORK

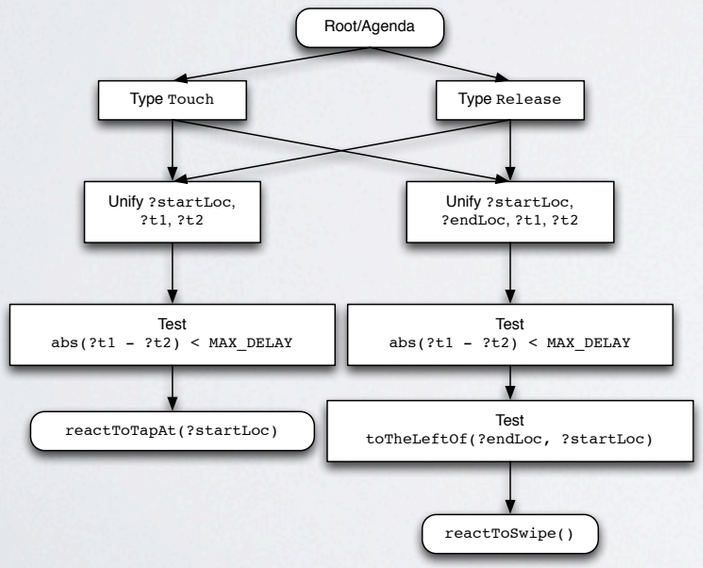
- **Parallel Production System**: K. Oflazer - 1985 - More state-saving
- **Parallel Rete**: Gupta et al. - 1986 - Quite fine-grained, but hardware-based scheduler
- **Treat**: D.P. Miranker - 1987 - Designed for specialized parallel hardware, less state-saving
- **PRAIS**: Goldstein et al. - 1991 - Blackboard, real-time guarantees
- **HOPES**: Dai et al. - 1992 - Blackboard, hierarchically organized
- **Lana**: Aref et al. - 1998 - Optimistic synchronization

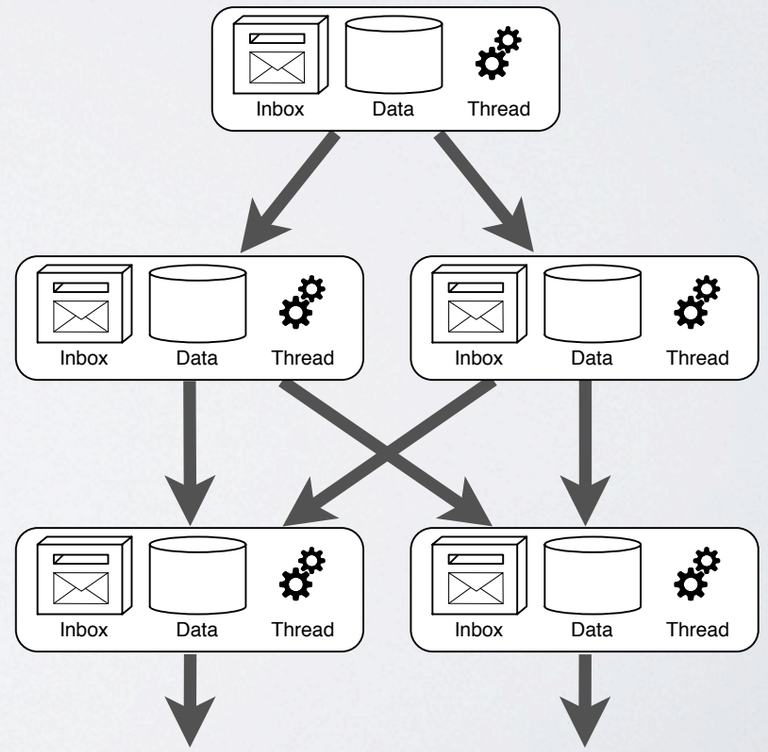
SUMMARY

- Problem:
  - Need for CED with guaranteed low latency



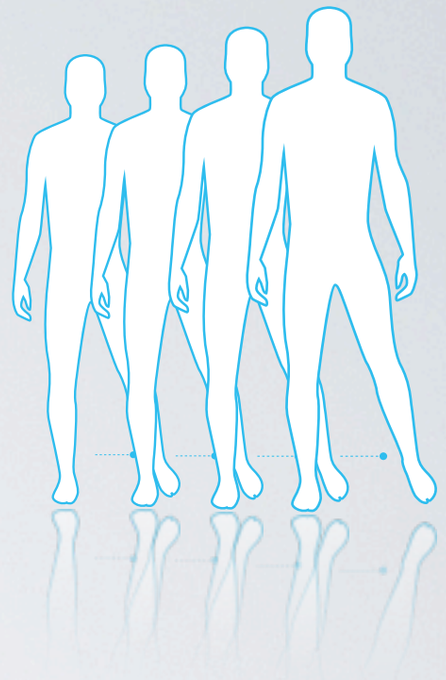
- Solution:
  - Actors with soft real-time guarantees



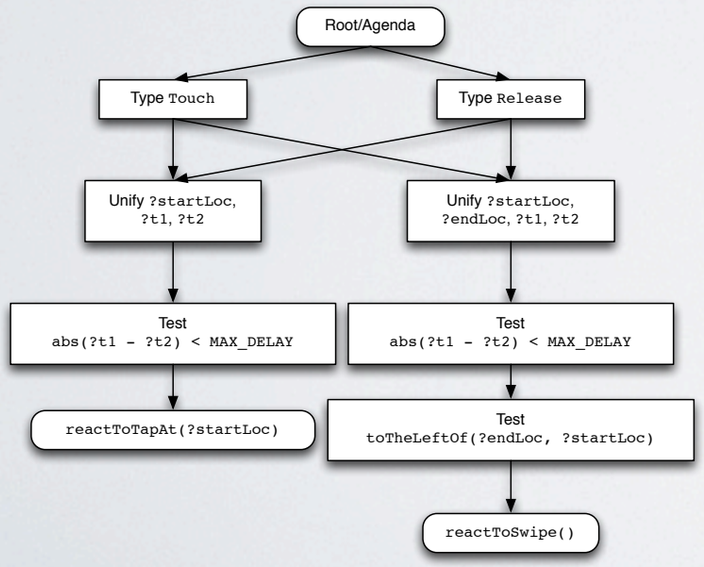


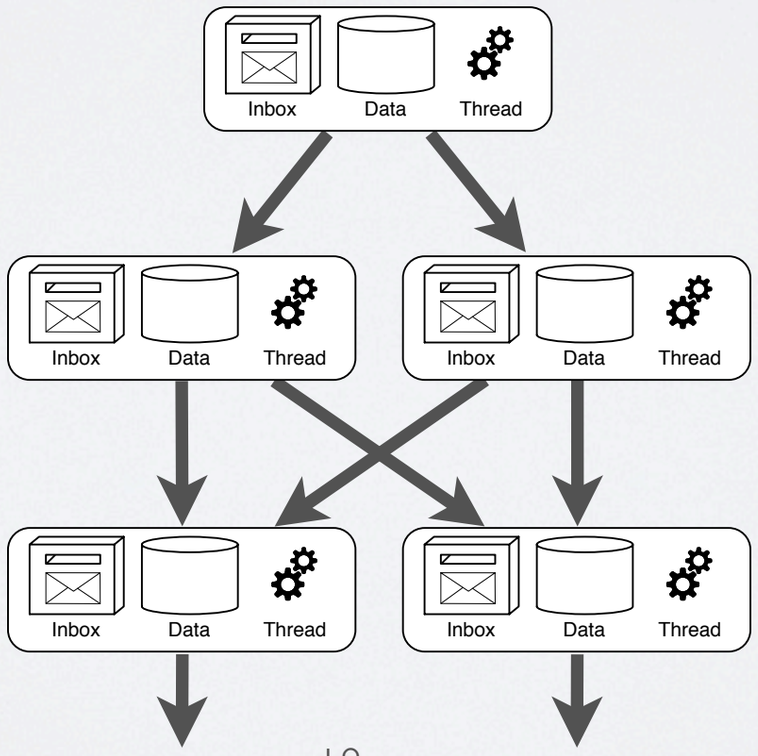
SUMMARY

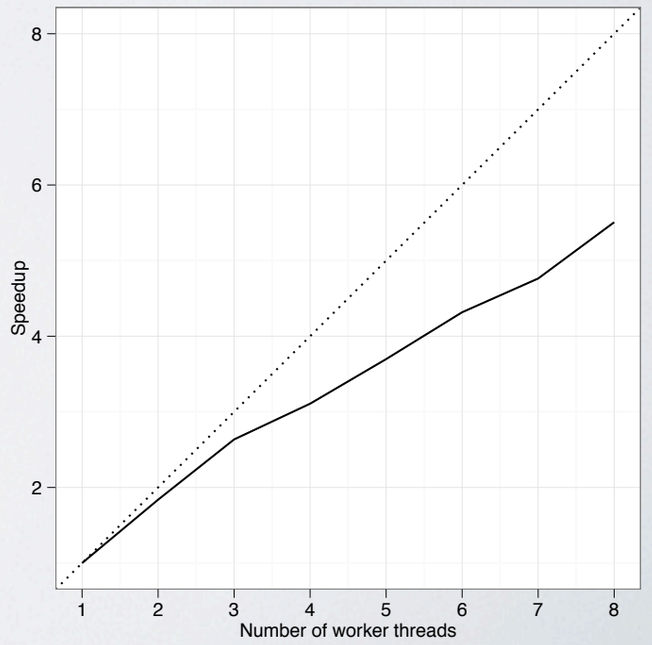
- Problem:
  - Need for CED with guaranteed low latency



- Solution:
  - Actors with soft real-time guarantees







PARALLEL GESTURE RECOGNITION WITH SOFT REAL-TIME GUARANTEES

Thierry Renaux
Lode Hoste
Stefan Marr
Wolfgang De Meuter

Software Languages Lab

Vrije Universiteit Brussel

Monday October 22nd, 2012 – AGERE! 2012