Profile Guided Offline Optimization of Hidden Class Graphs for JavaScript VMs in Embedded Systems

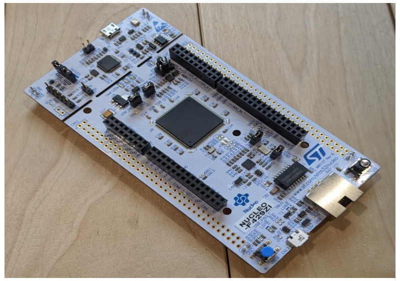
Tomoharu Ugawa, University of Tokyo

Stefan Marr, University of Kent

Richard Jones, University of Kent

JavaScript in IoT

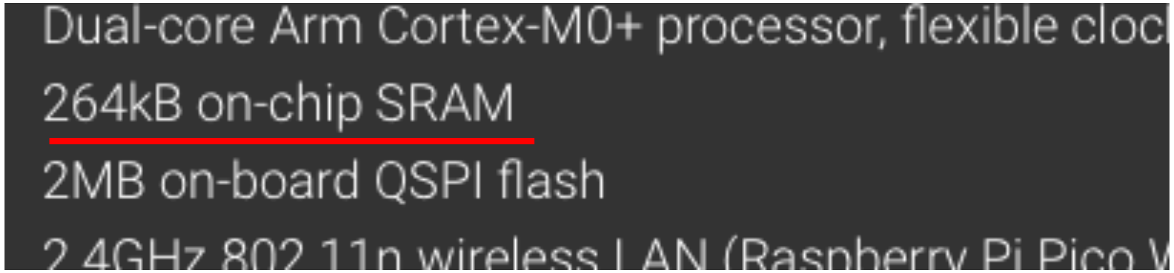
- JavaScript engines for IoT became popular
  - IoT.js, Moddable, eJSVM,…
- Challenge: memory footprint
  - Around 256 KB of RAM is available
  - More than 20 KB of RAM is occupied by meta-objects in eJSVM



STM32F429

- Arm Cortex-M4
- **256 KB of SRAM**

Raspberry Pi pico specification



Dual-core Arm Cortex-M0+ processor, flexible cloc

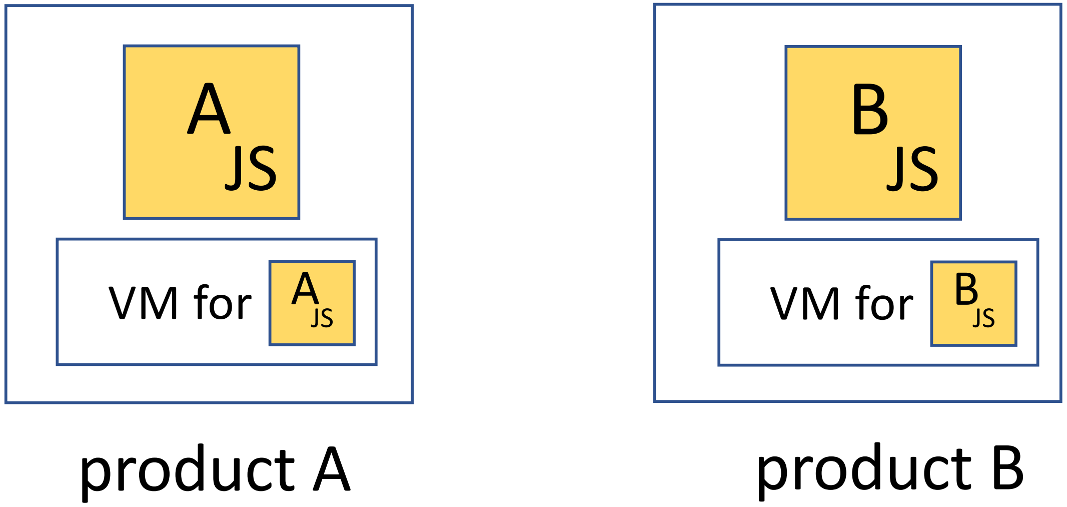
264kB on-chip SRAM

2MB on-board QSPI flash

2.4GHz 802.11n wireless LAN (Raspberry Pi Pico W

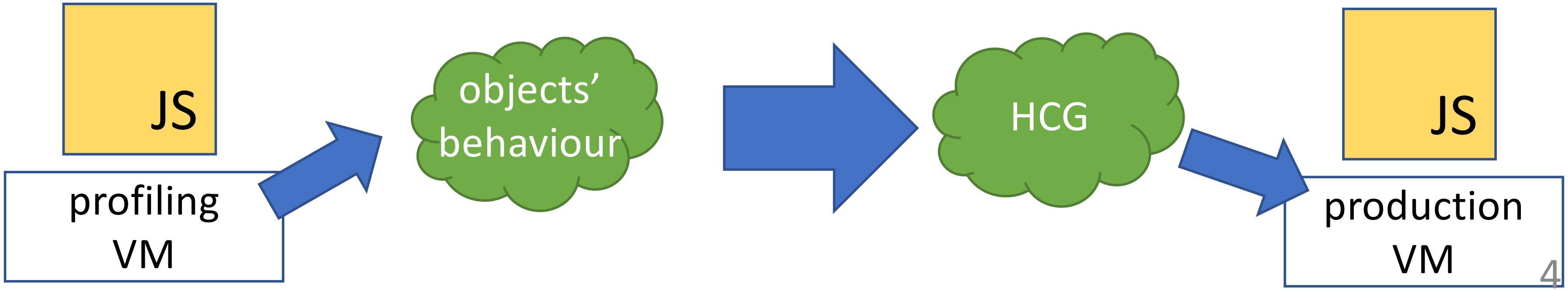
Closed World Assumption

- We can assume program is fixed for a particular IoT product
  - For product A, VM executes only A.js
- VM specialisation to a particular application is feasible



Overview of Our Work

- Specialise hidden class graph (HCG)
  - HCG represents type information of objects
  - HCG is created and grows during execution in accordance with program's behaviour
- Steps
  1. Collect objects' behaviour from profiling run
  2. Construct a static HCG and optimise it offline
  3. Use static HCG in actual runs



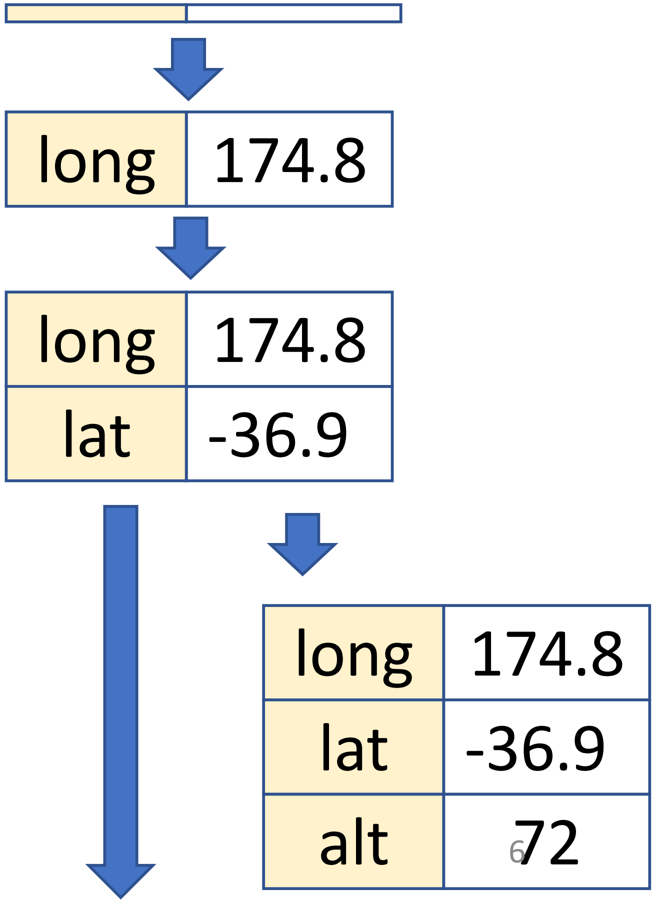
Agenda

- Introduction
- Hidden classes
- Optimised Hidden Class Construction
- Evaluation

JavaScript Object

- Not statically typed
  - Properties are added dynamically
  - Set of properties depends on control-flow

```
readGPS() {
  let loc = {};
  loc.long = getLongitude();
  loc.lat = getLatitude();
  if (hasAltitude())
    loc.alt = getAltitude();
  return loc;
}
```



long	174.8

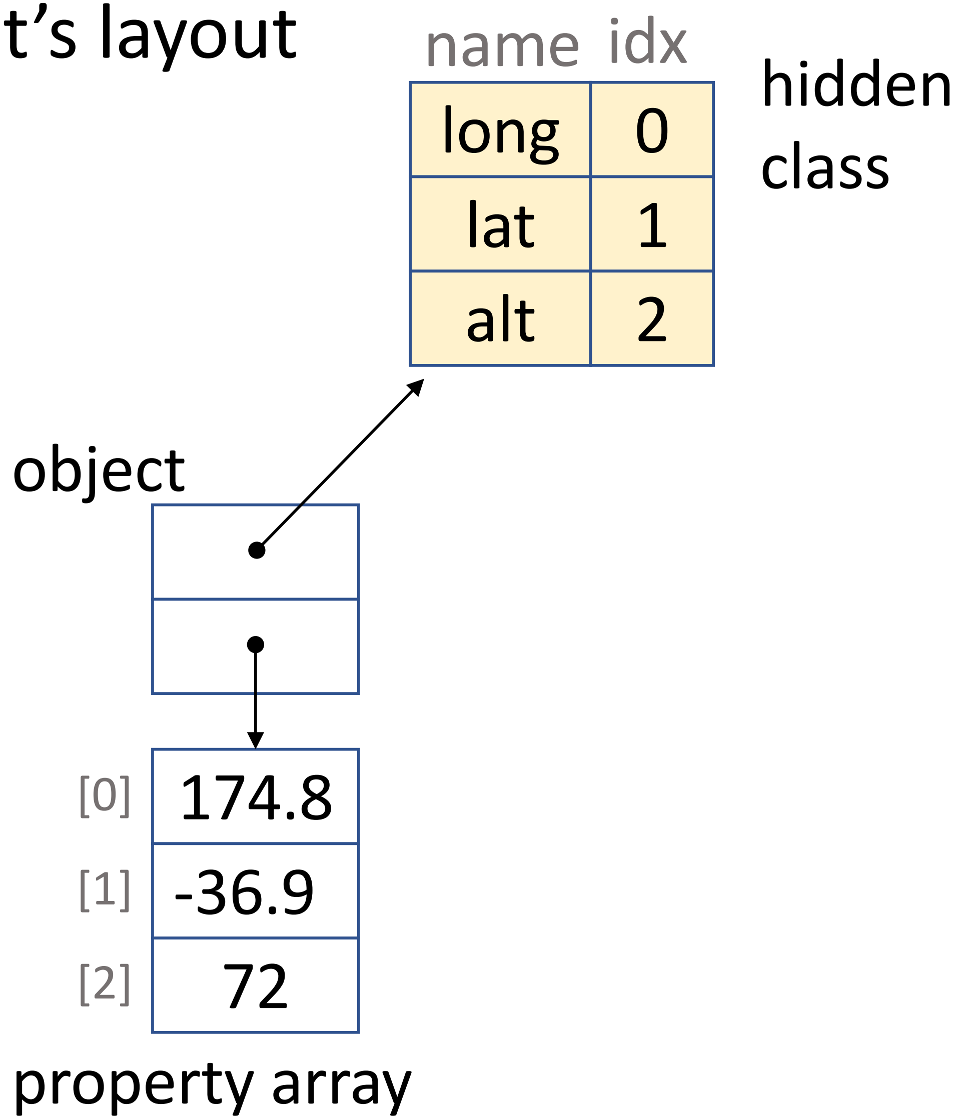
long	174.8
lat	-36.9

long	174.8
lat	-36.9
alt	72

Hidden Class (HC)

- Meta-object having object's layout
- object = (HC, prop array)

long	174.8
lat	-36.9
alt	72



hidden class

name	idx
long	0
lat	1
alt	2

object

property array

[0]	174.8
[1]	-36.9
[2]	72

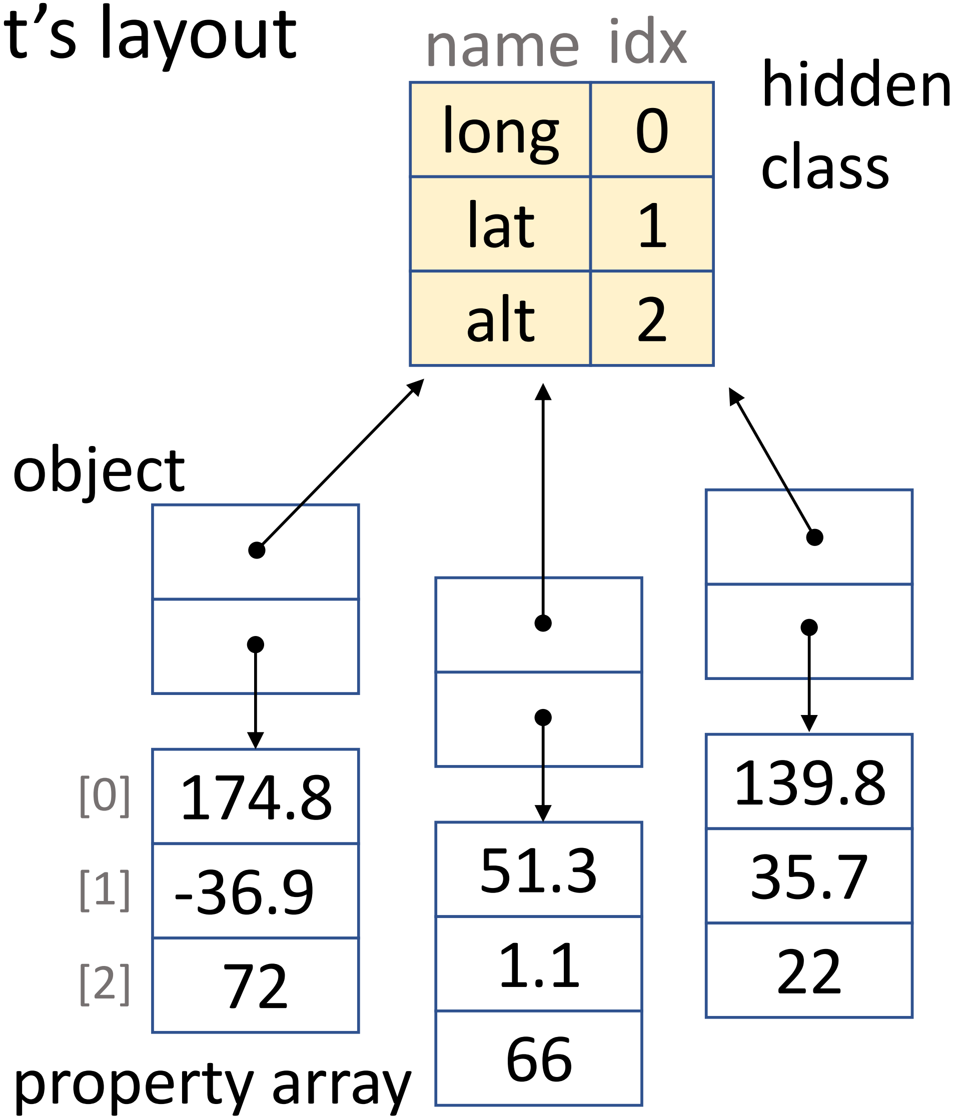
Hidden Class (HC)

- Meta-object having object’s layout
- object = (HC, prop array)
- Shared with all instances

long	174.8
lat	-36.9
alt	72

long	174.8
lat	35.7
alt	40

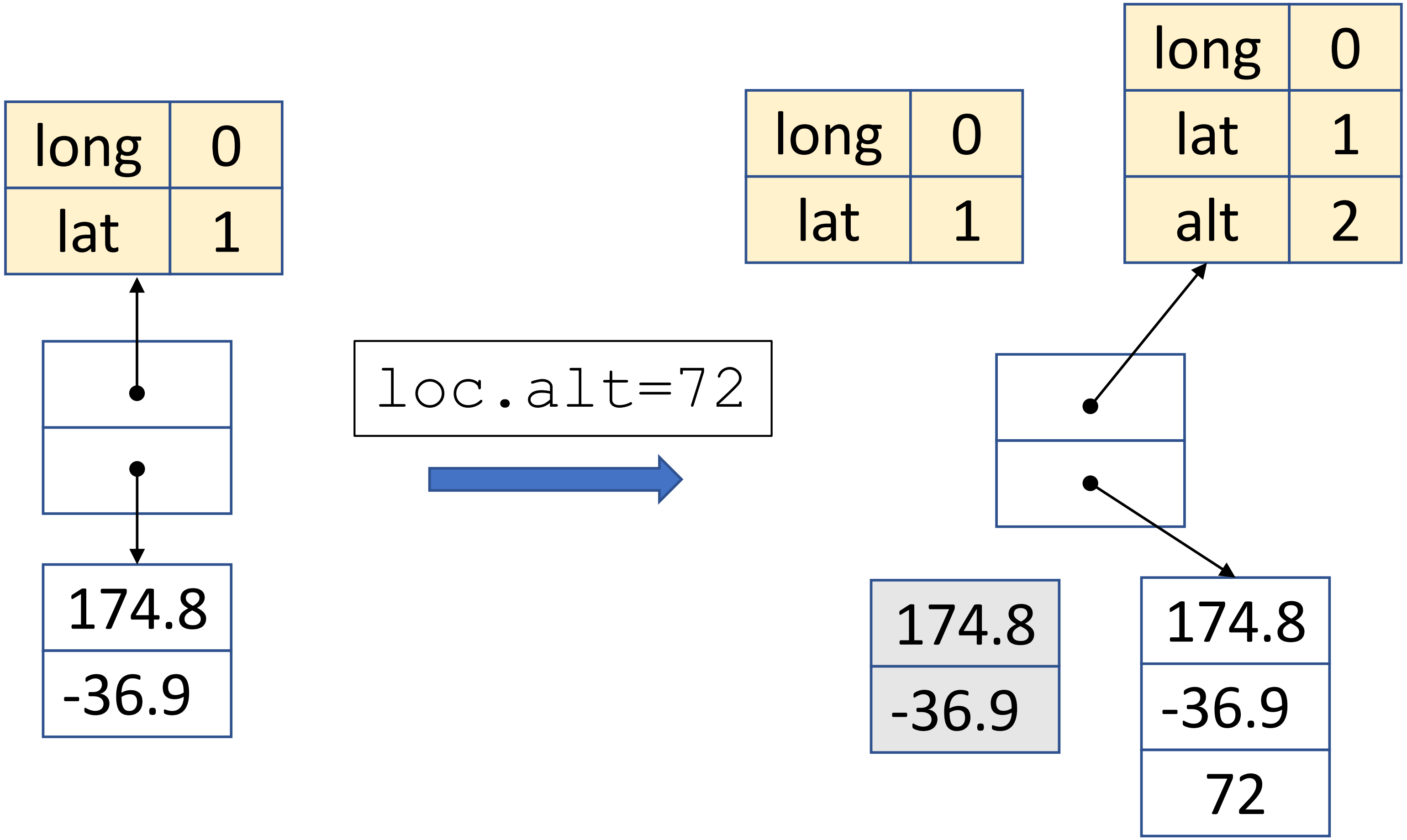
long	51.3
lat	1.1
alt	66



name	idx
long	0
lat	1
alt	2

Hidden Class Transition

- Adding new property causes HC transition
  - Find next HC, or create it if it has not been created
  - Re-allocate property array



long	0
lat	1

```
loc.alt=72
```

long	0
lat	1

long	0
lat	1
alt	2

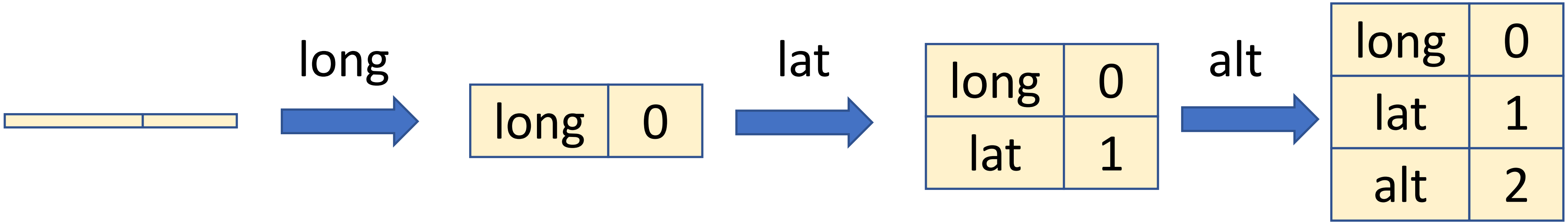
174.8
-36.9

174.8
-36.9

174.8
-36.9
72

Hidden Class Graph

- Hidden class graph (HCG) enables to find next HC quickly
    - node: HC
    - edge: transition labelled with property name



long

long	0

lat

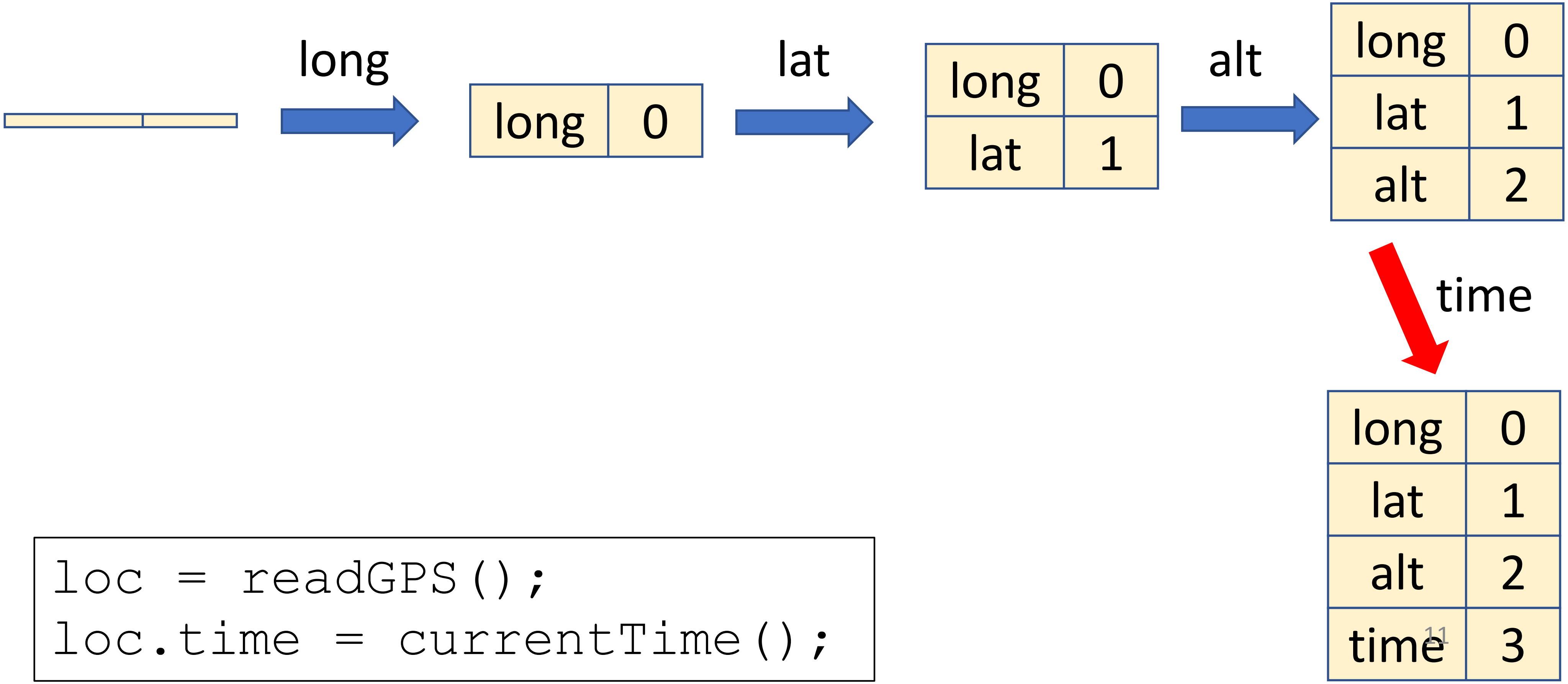
long	0
lat	1

alt

long	0
lat	1
alt	2

HCG grows during execution

- New property creates new HC
- New HC is added to HCG



long	0

long	0
lat	1

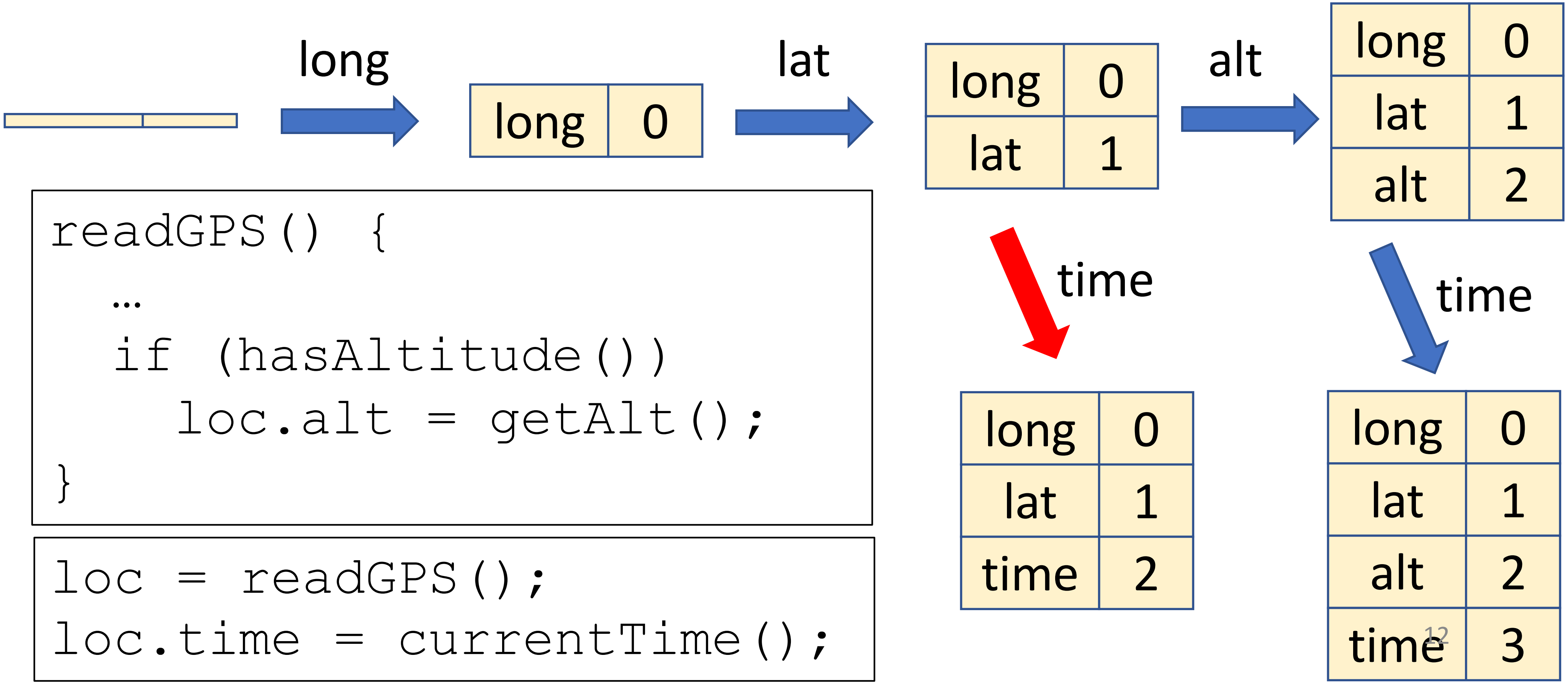
long	0
lat	1
alt	2

long	0
lat	1
alt	2
time	3

```
loc = readGPS();
loc.time = currentTime();
```

HCG grows during execution

- New property creates new HC
- New HC is added to HCG



```
readGPS() {
  …
  if (hasAltitude())
    loc.alt = getAlt();
}
```

```
loc = readGPS();
loc.time = currentTime();
```

long	0

long	0
lat	1

long	0
lat	1
alt	2

long	0
lat	1
time	2

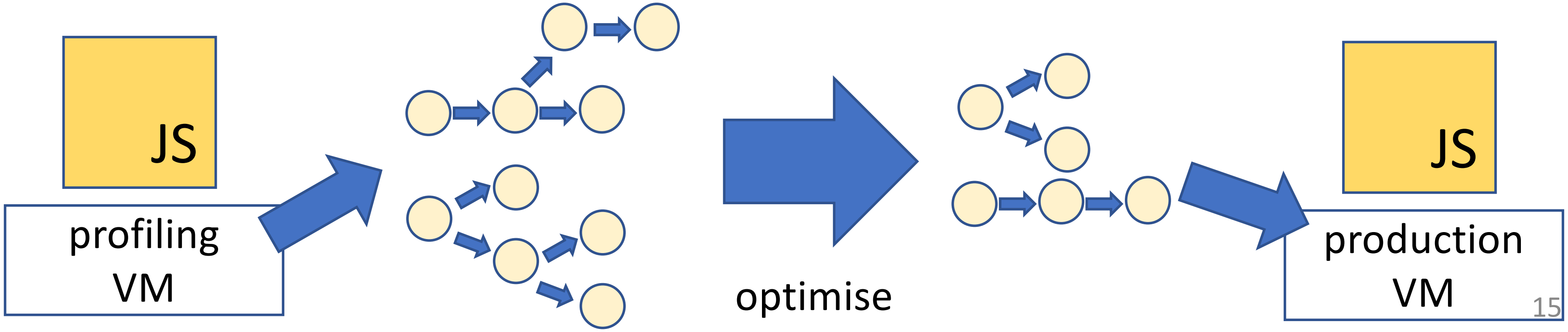
long	0
lat	1
alt	2
time	3

Agenda

- Introduction
- Hidden classes
- Optimised Hidden Class Construction
- Evaluation

Offline Optimisation of HCG

- Optimise HCG with the following policy
  - Reduce memory footprint
    - Shrink HCG and reduce object size
  - Allow small space-inefficiency for speed
- Use optimised HCG in production VM
  - Run-time optimisation relying on assumption that HCG is stable

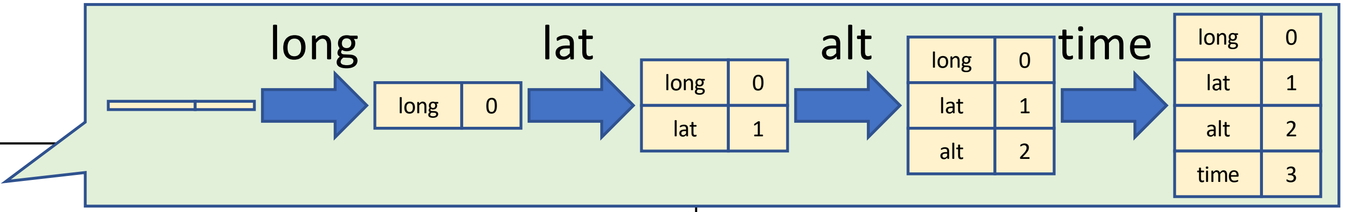


Optimisations

1. Eliminating intermediate HCs
2. Moving branches

Layout-monomorphic allocation site

- 95.8 % of allocation sites are layout-monomorphic
- Layout-monomorphic allocation site:
  all objects allocated there obtain the same set of properties in the same order
  - Eventually get transitions to the same HC



long

long	0

lat

long	0
lat	1

alt

long	0
lat	1
alt	2

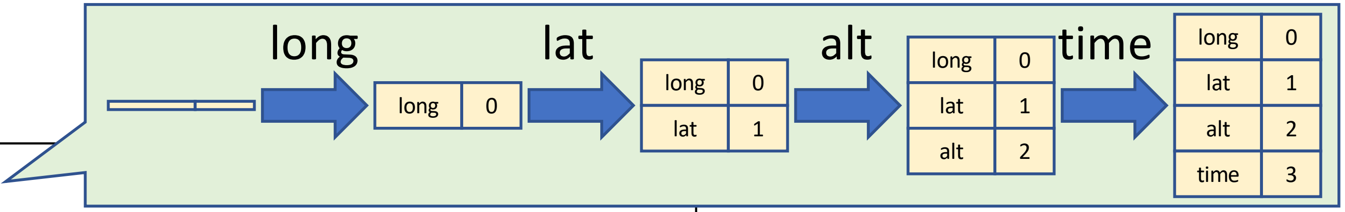
time

long	0
lat	1
alt	2
time	3

```
let loc = {};
loc.long = getLongitude();
loc.lat  = getLatitude();
loc.alt  = getAltitude();
loc.time = getCurrentTime();
```

Optimisation for layout-monomorphism: pre-transitioning

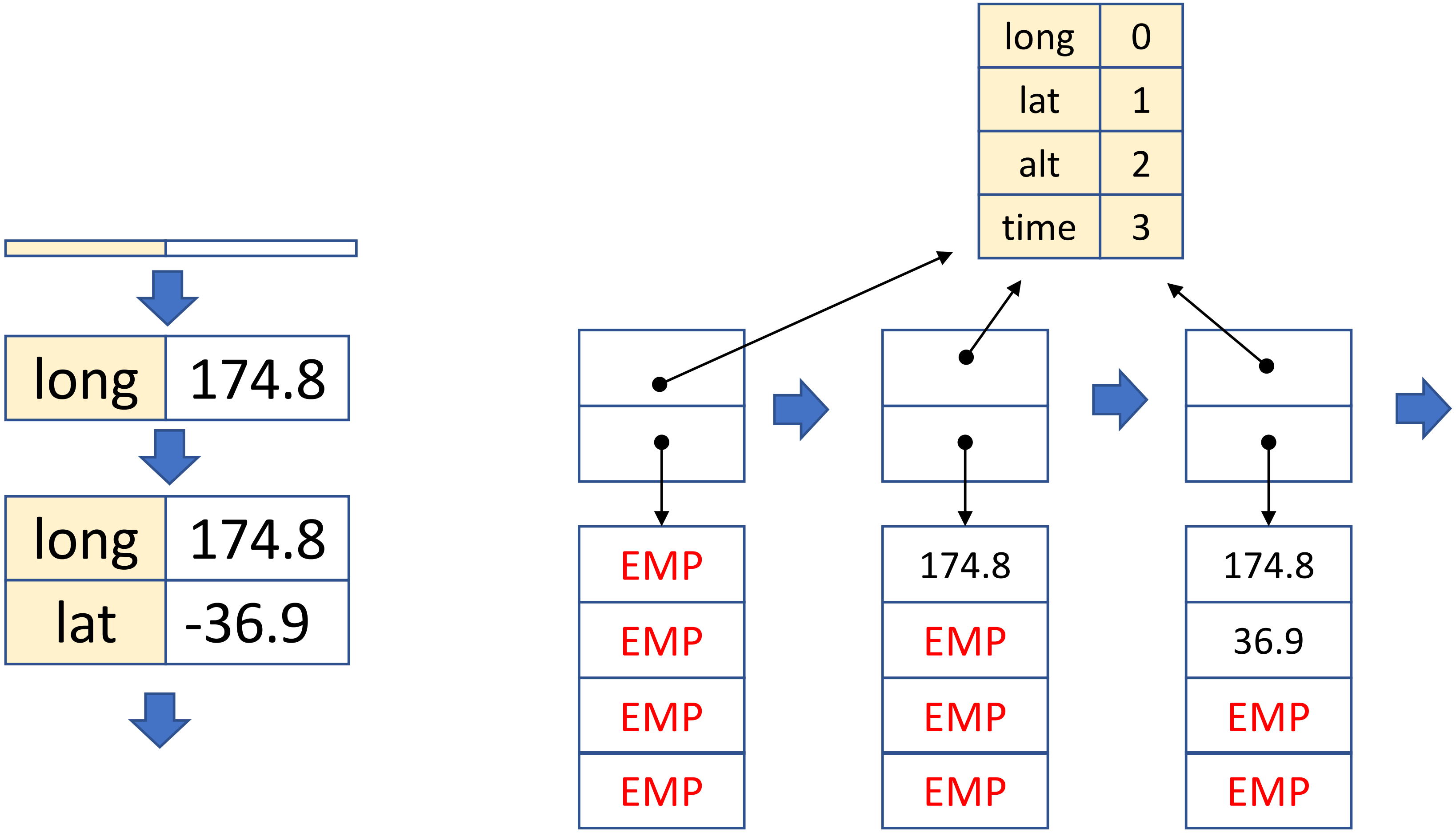
- Eliminate all hidden classes but the last from HCG
- Objects are created with their final layout
  - No re-allocation overhead of property array



```
let loc = {};
loc.long = getLongitude();
loc.lat  = getLatitude();
loc.alt  = getAltitude();
loc.time = getCurrentTime();
```

EMPTY value

- Initialise property array slots with EMP to indicate absence of the property
  - Allow us to search for property in the prototype object.



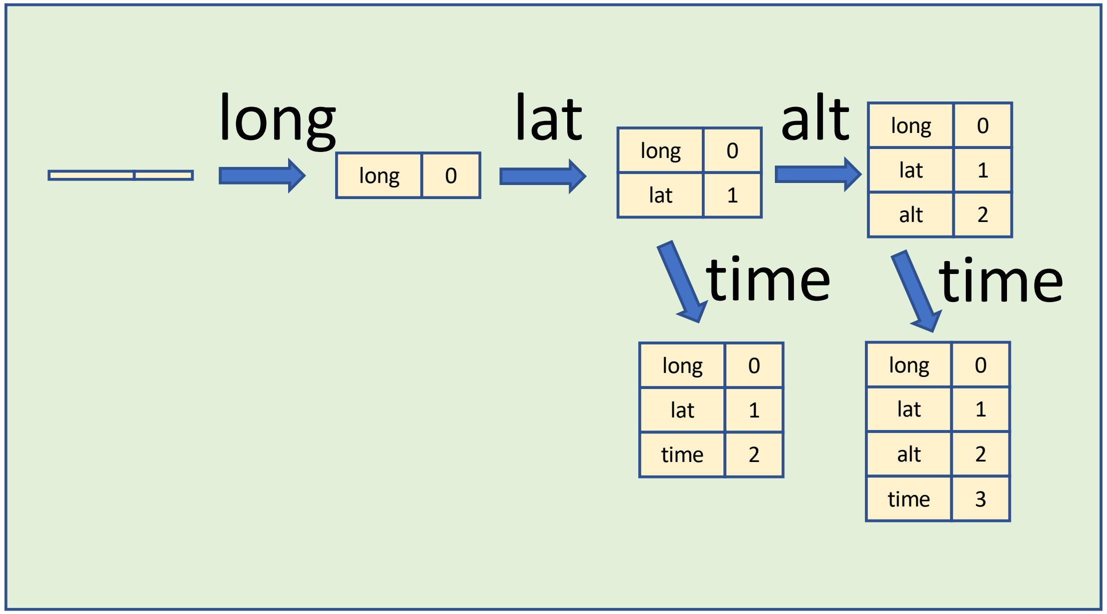
long	0
lat	1
alt	2
time	3

long	174.8

long	174.8
lat	-36.9

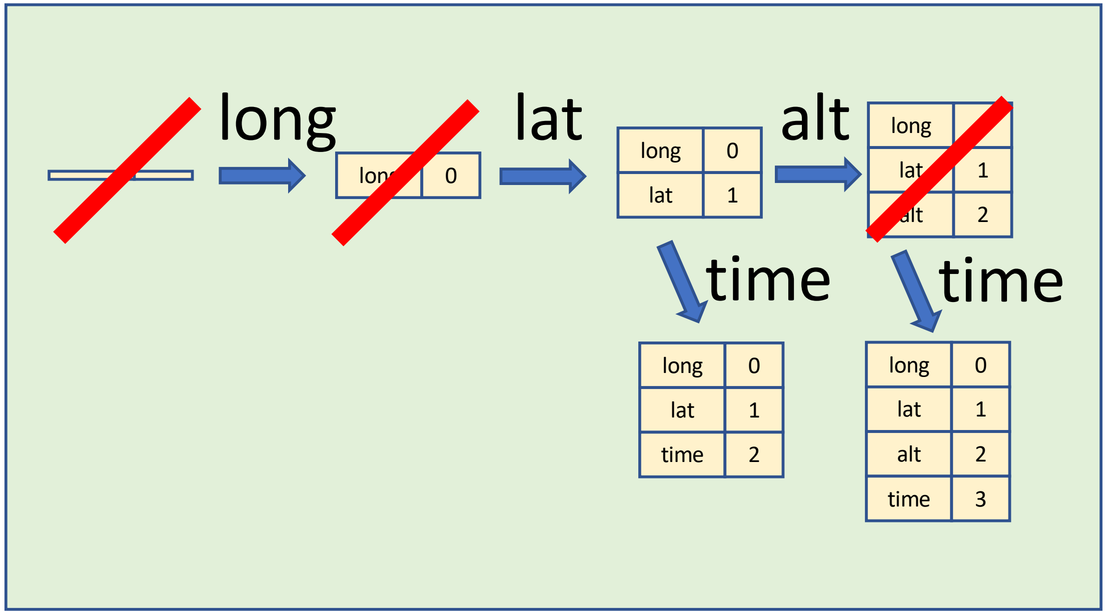
Optimisation 1: elimination of Intermediate HCs

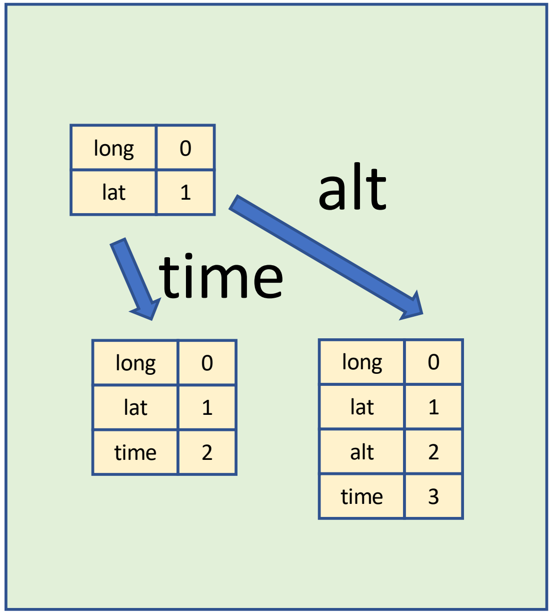
- Generalization of pre-transitioning
- Eliminate all internal nodes but branching nodes



Optimisation 1: elimination of Intermediate HCs

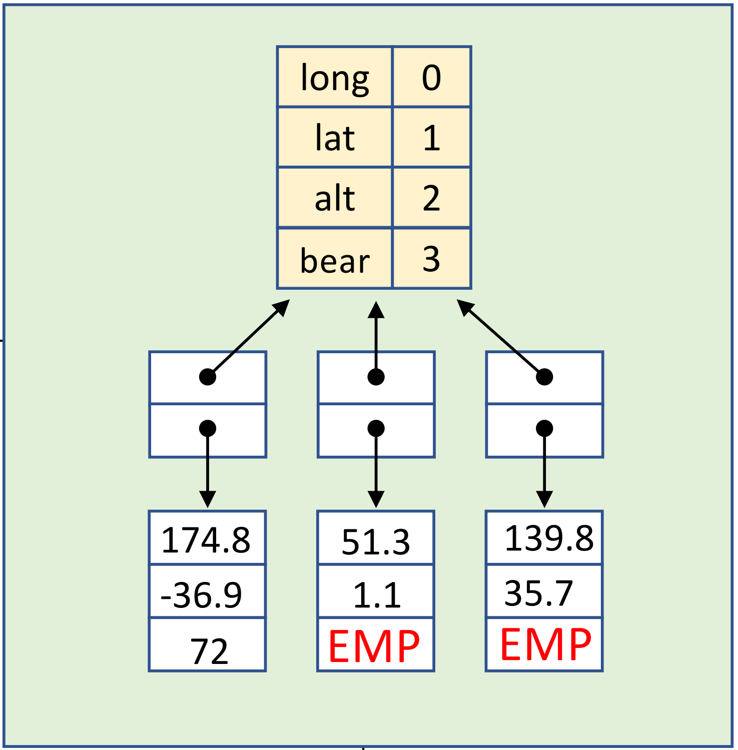
- Generalization of pre-transitioning
- Eliminate all internal nodes but branching nodes





Over-allocation

- Aggressive elimination increases memory footprint
  - Memory for all possible properties are reserved
- Examples
  - Parts of objects get extra props
  - Props are added in the future

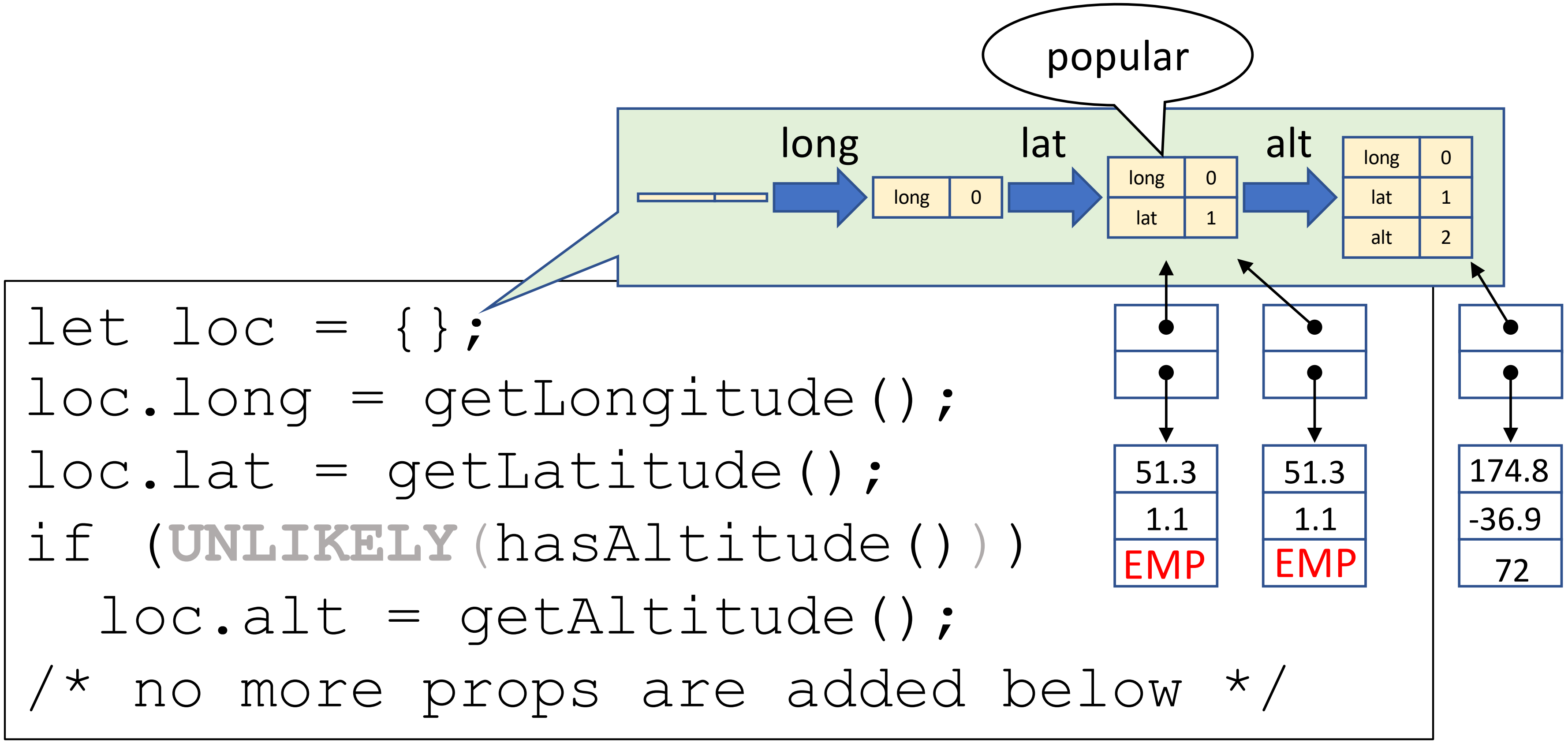


```
let loc = {};
loc.long = getLongitude();
loc.lat = getLatitude();
if (UNLIKELY(hasAltitude()))
  loc.alt = getAltitude();
/* no more props are added below */
```

Optimisation 1’: preserve popular HCs

- popular HC: $\max_{t \in execution} (\#\text{of instances}) > K$
  - Sample # of instances at each GC cycle

$K = 10$

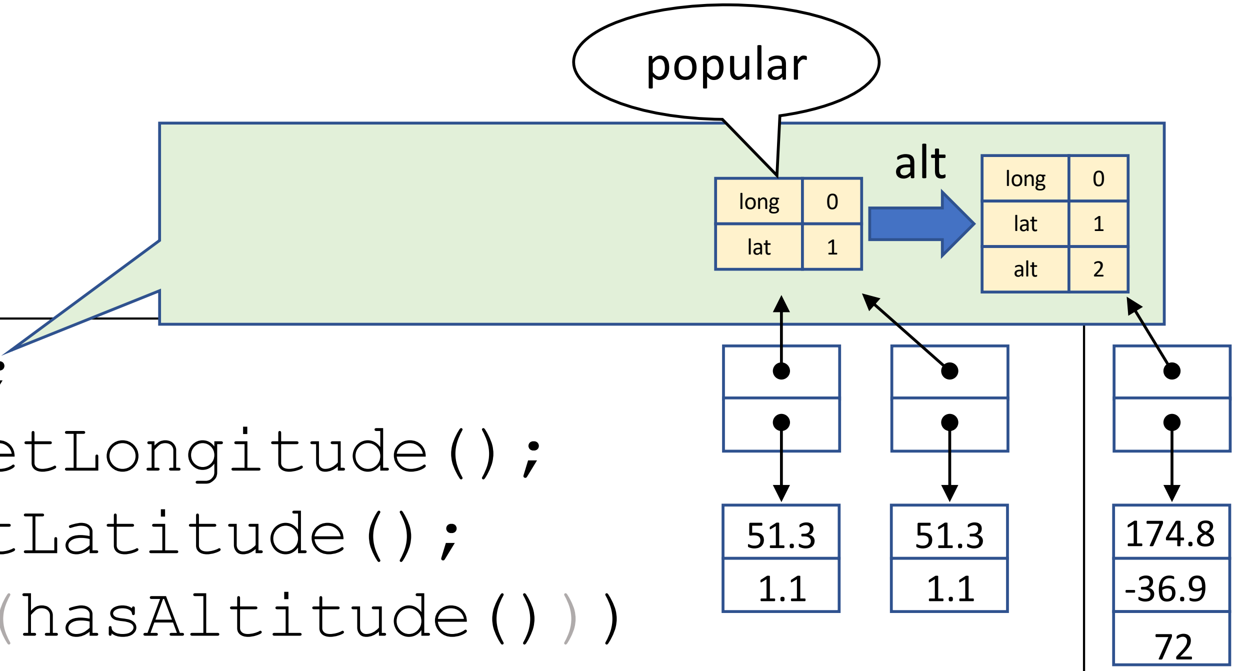


```
let loc = {};
loc.long = getLongitude();
loc.lat = getLatitude();
if (UNLIKELY(hasAltitude()))
  loc.alt = getAltitude();
/* no more props are added below */
```

Optimisation 1’: preserve popular HCs

- popular HC: $\max_{t \in execution}(\#\text{of instances}) > K$
  - Sample # of instances at each GC cycle

K = 10

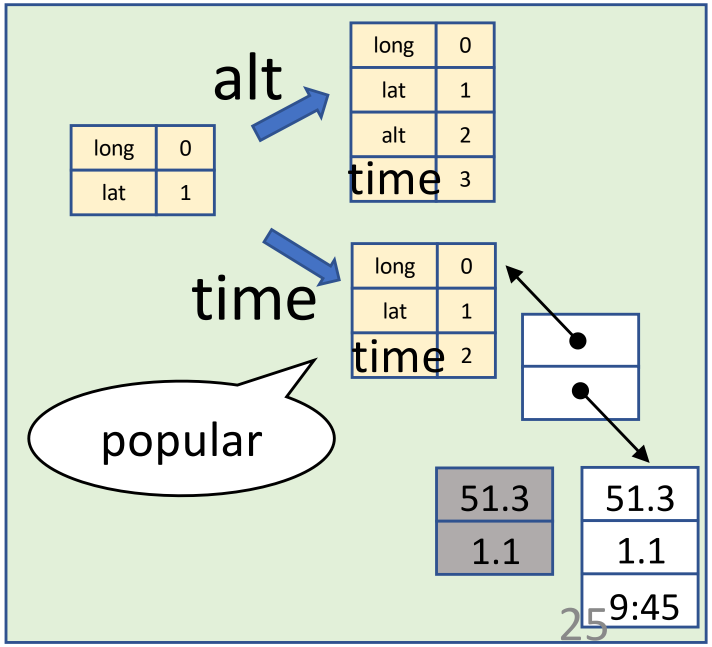


```
let loc = {};
loc.long = getLongitude();
loc.lat = getLatitude();
if (UNLIKELY(hasAltitude()))
  loc.alt = getAltitude();
/* no more props are added below */
```

Motivating example for optimisation 2

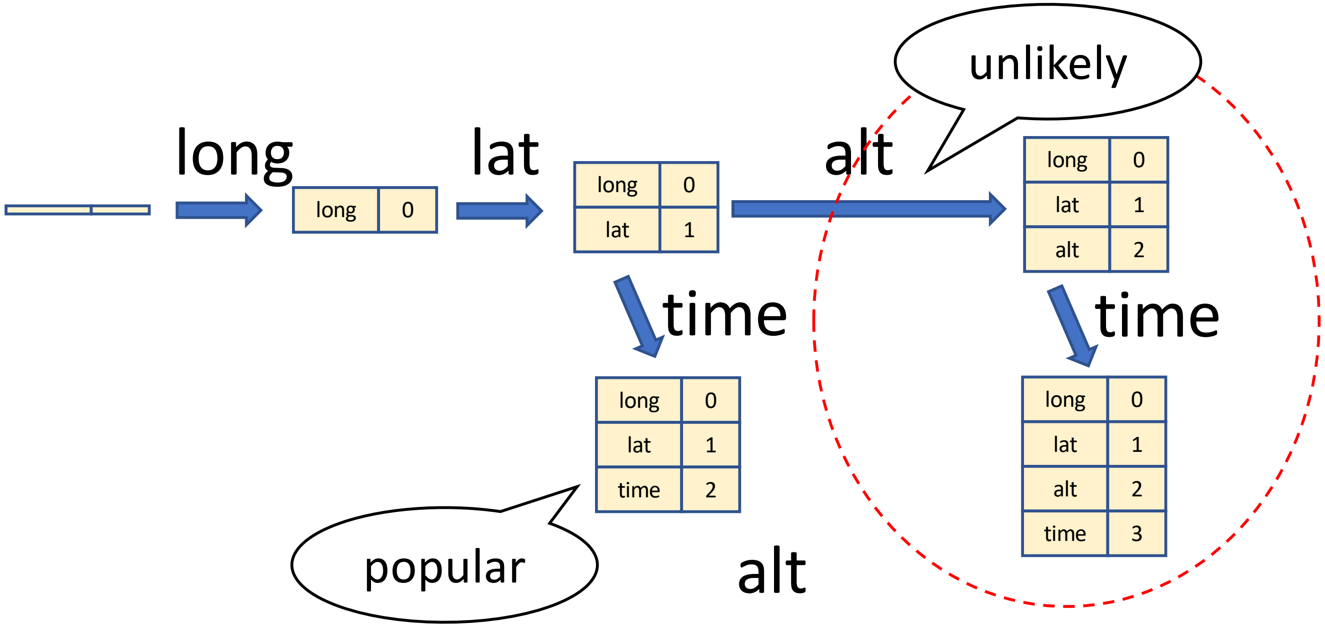
- Common properties are added after a branch
- Every object experiences property array re-allocation

```
let loc = {};
loc.long = getLongitude();
loc.lat = getLatitude();
if (UNLIKELY(hasAltitude()))
  loc.alt = getAltitude();
loc.time = currentTime();
```



Optimisation 2: moving branches

- Move “unlikely” branch before optimization 1



long	0

long	0
lat	1

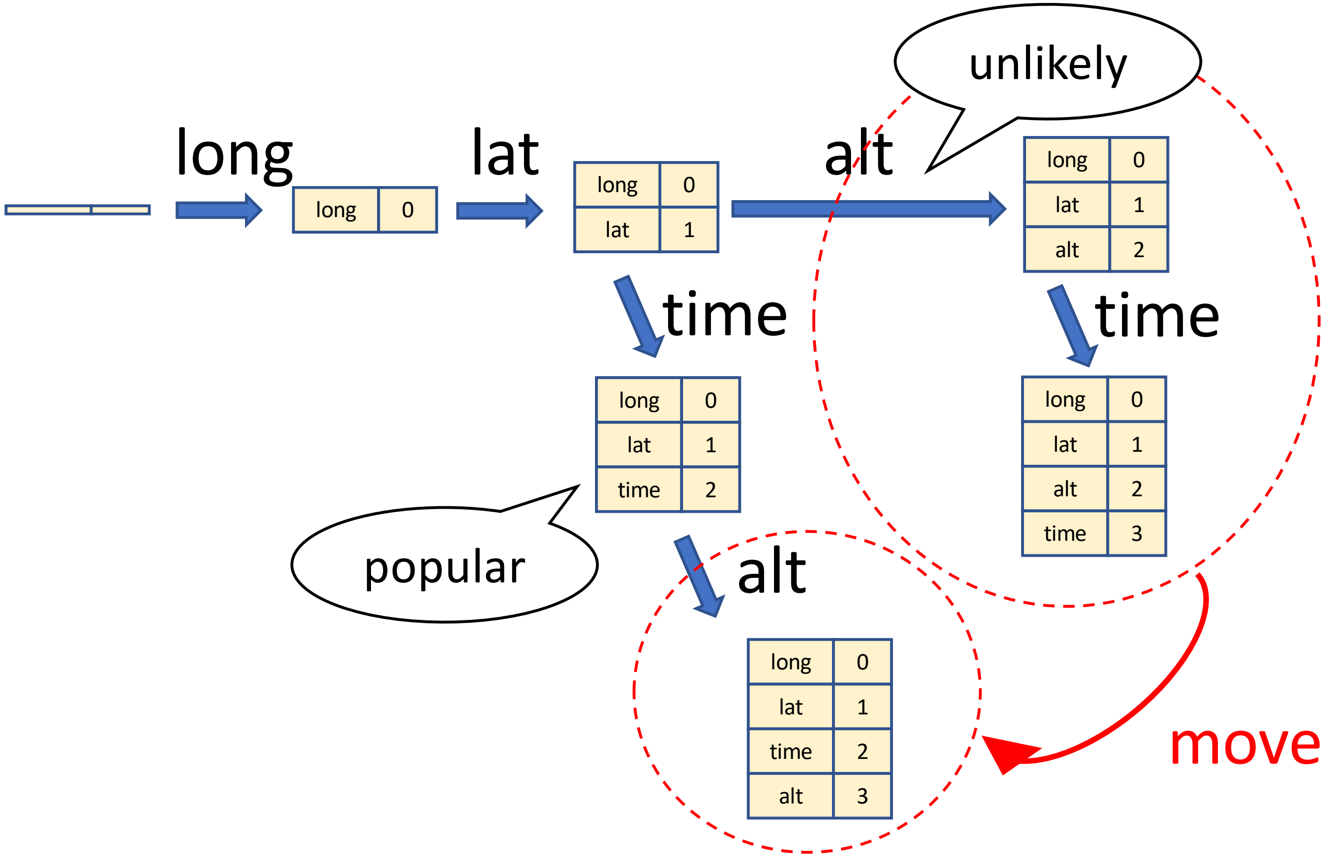
long	0
lat	1
alt	2

long	0
lat	1
time	2

long	0
lat	1
alt	2
time	3

Optimisation 2: moving branches

- Move “unlikely” branch before optimization 1



long	0

long	0
lat	1

long	0
lat	1
alt	2

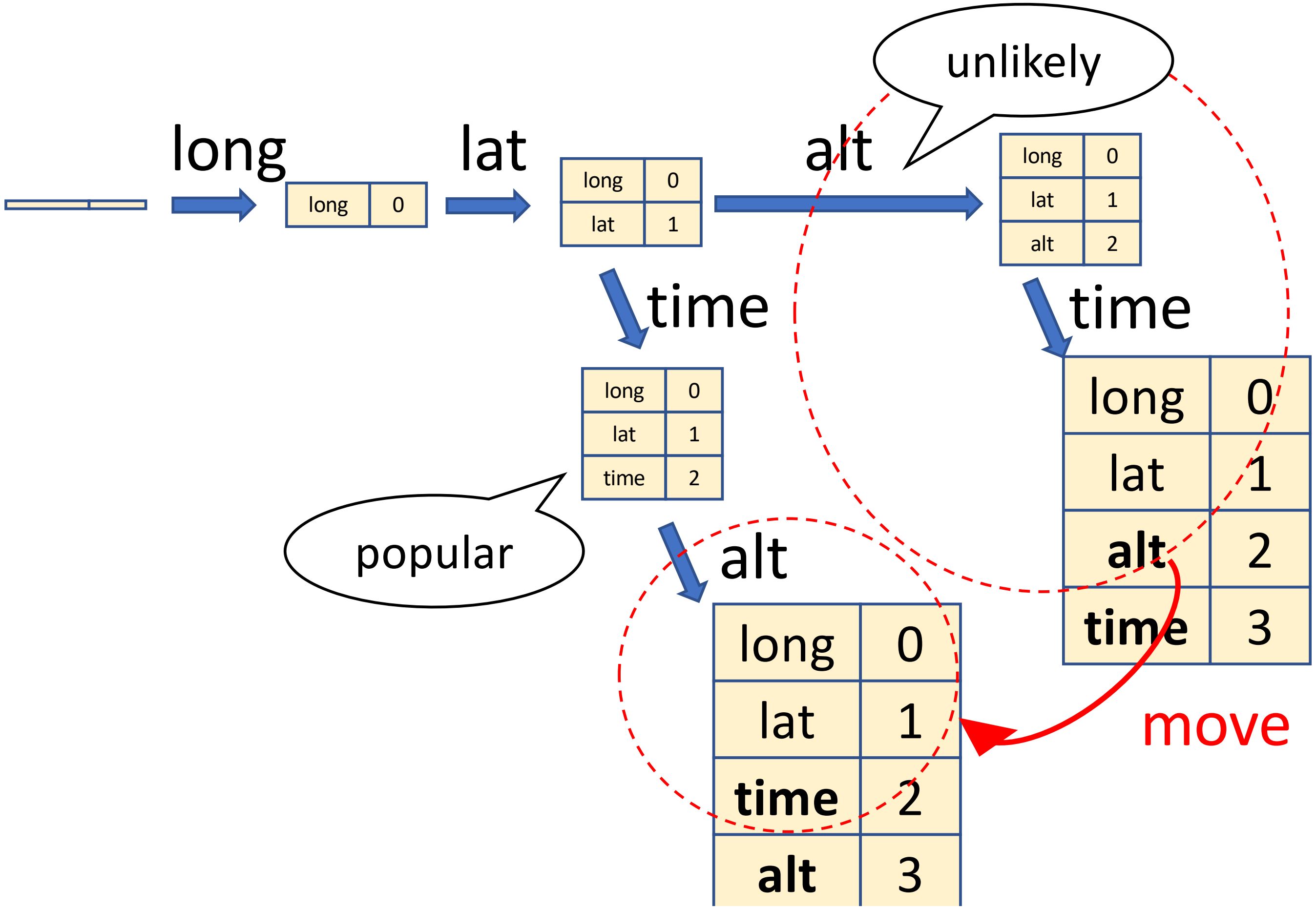
long	0
lat	1
time	2

long	0
lat	1
alt	2
time	3

long	0
lat	1
time	2
alt	3

Optimisation 2: moving branches

- Move “unlikely” branch before optimization 1



long	0

long	0
lat	1

long	0
lat	1
alt	2

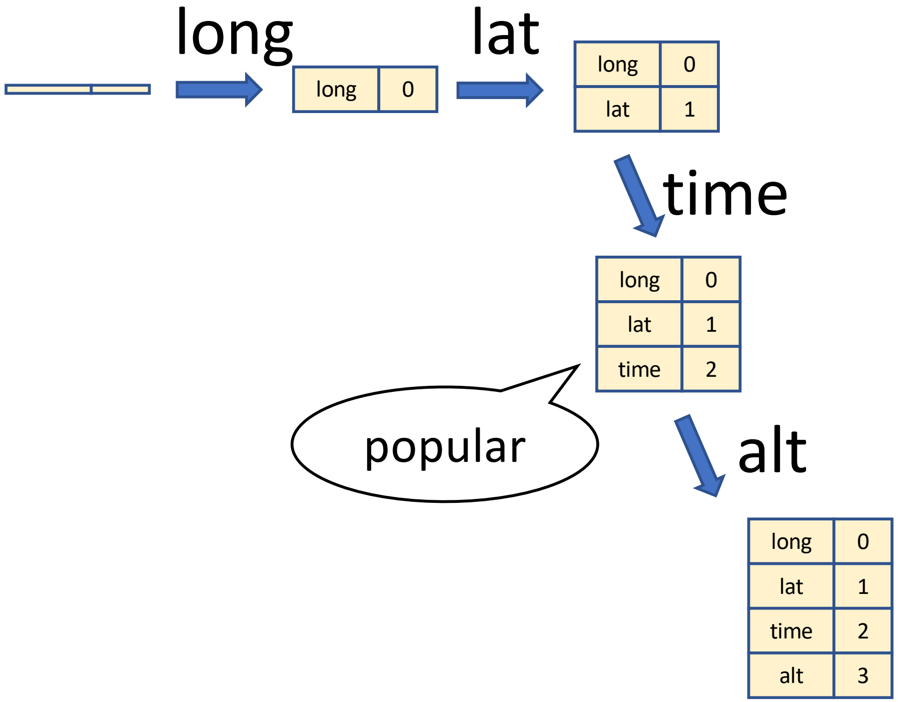
long	0
lat	1
time	2

long	0
lat	1
alt	2
time	3

long	0
lat	1
time	2
alt	3

Optimisation 2: moving branches

- Move “unlikely” branch before optimization 1
  - Linearise HCG



long

long	0

lat

long	0
lat	1

time

long	0
lat	1
time	2

popular

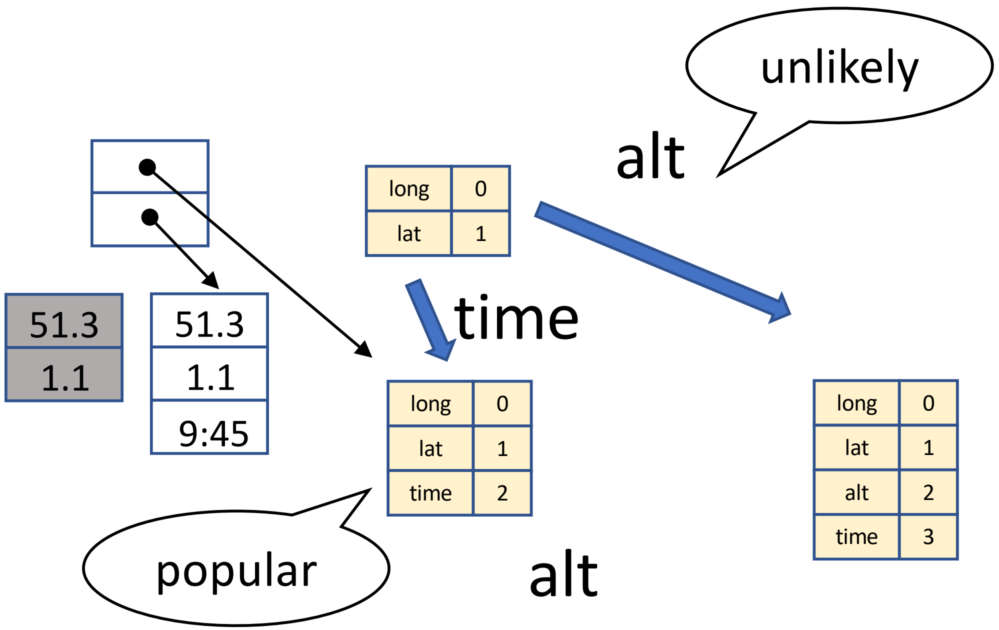
alt

long	0
lat	1
time	2
alt	3

Benefits of moving branches

- Encourage elimination of intermediate HCs
- Majority of objects are created with final layout

without moving branches



51.3	
1.1	

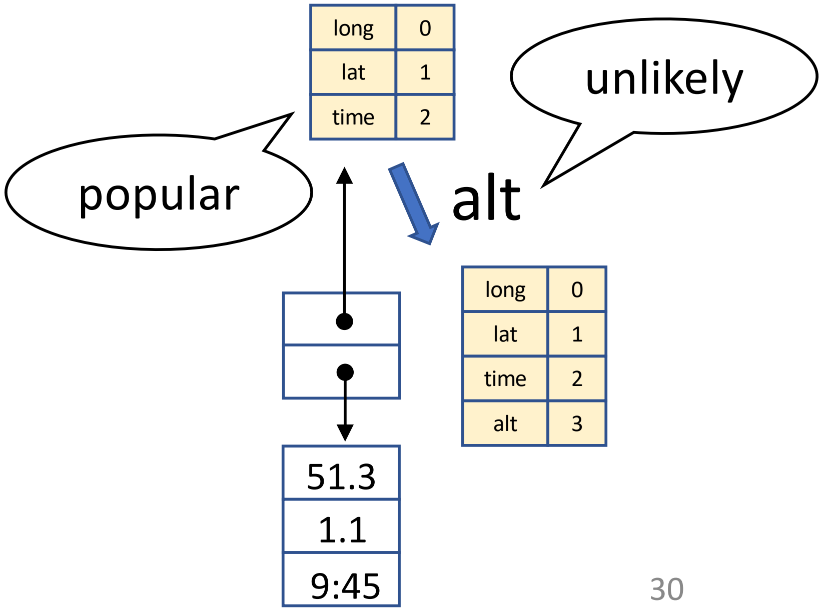
51.3	
1.1	
9:45	

long	0
lat	1

long	0
lat	1
time	2

long	0
lat	1
alt	2
time	3

with moving branches



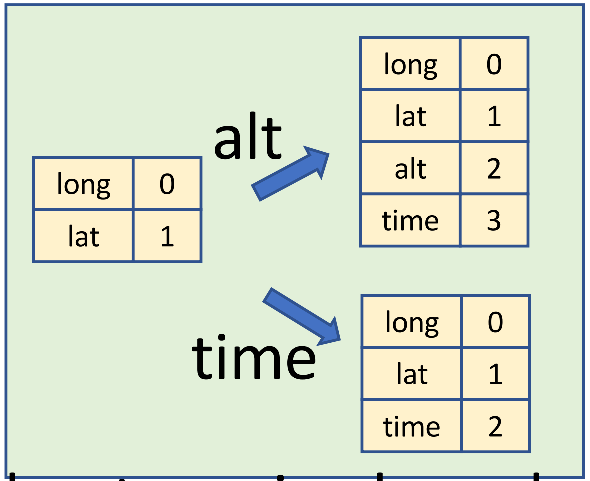
long	0
lat	1
time	2

long	0
lat	1
time	2
alt	3

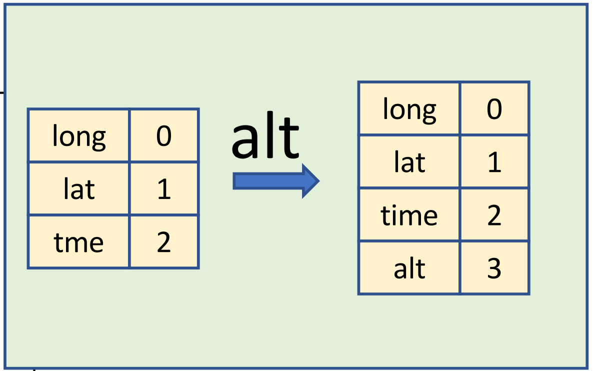
51.3
1.1
9:45

Benefits of moving branches to inline cache

- Moving branch reduces variations of HCs
- Improves inline cache hit ratio
  - inline cache gives index if object has the same HC as cached



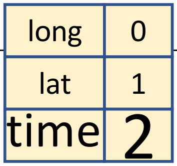
without moving branches



with moving branches

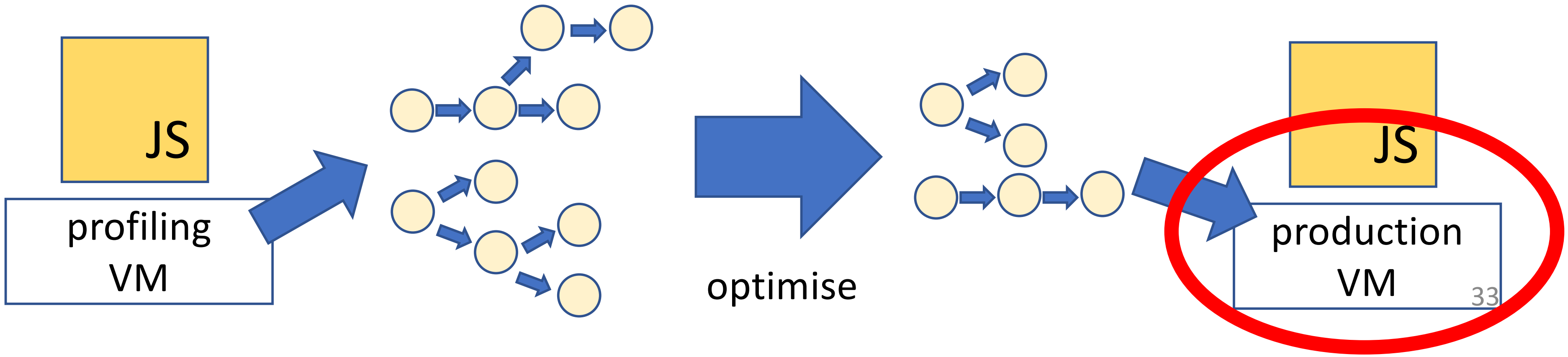
```
localTime(loc) {
  tdiff = floor(loc.long / 15);
  return loc.time + tdiff;
}
```

```
if (obj->HC ==           )
  return obj->props[2];
else
  slow_path();
```



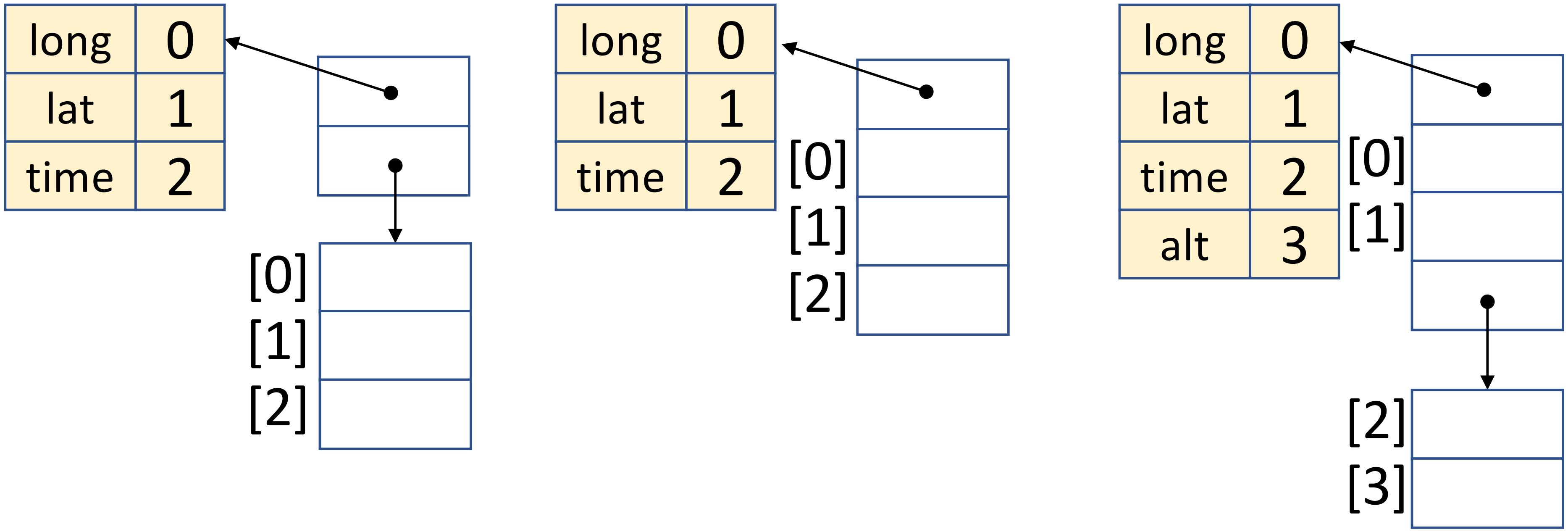
Run-time optimisation

- Run-time optimisation relying on assumption that HCG is stable
  - in-object allocation
  - baking HCG into flash memory (future work)



in-object allocation

- Allocate all properties *in* object
  - Save space for indirect pointer
- In case of overflow,
  convert the last property area to indirect pointer

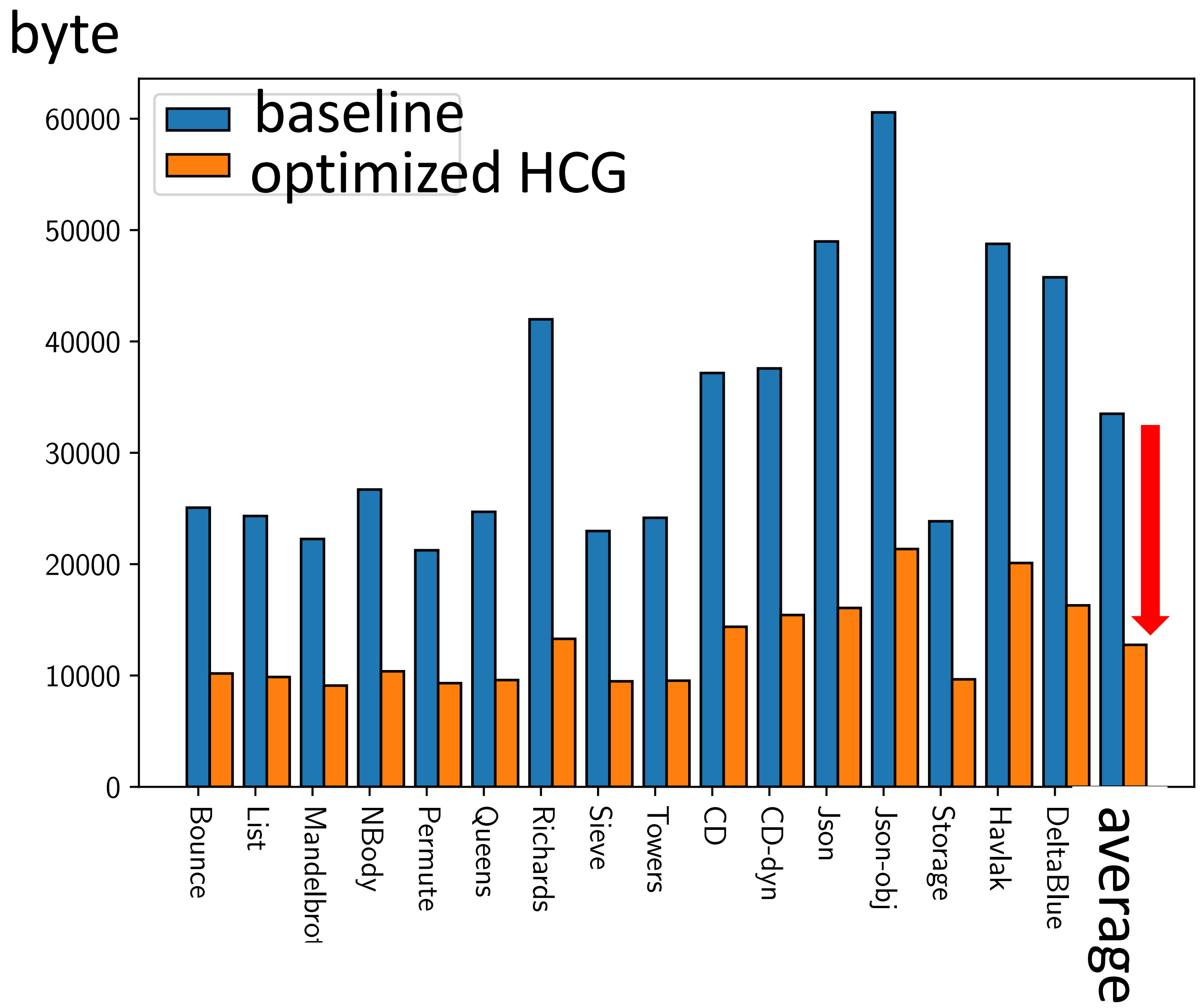


original version   in-object allocation   overflow

Evaluation

- Implemented in eJSVM
- Are we fast yet benchmarks
  - original benchmarks
  - JSON-obj: uses an AST node object as a dictionary
  - CD-dyn: do not initialise future properties with NULL

Size of HC-related data

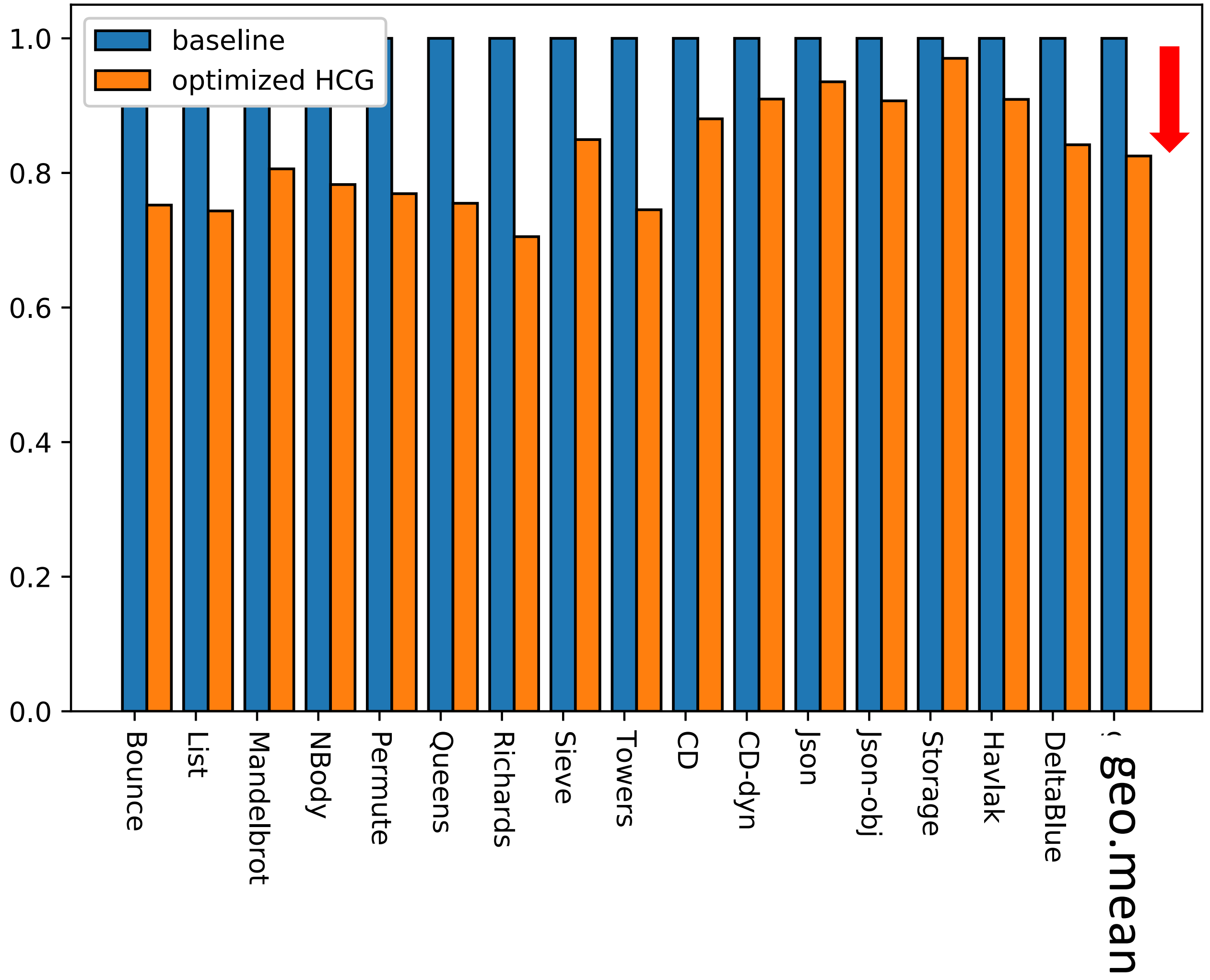


reduced by 61.9%
from 33.5 KB
to 12.8KB

nomadised

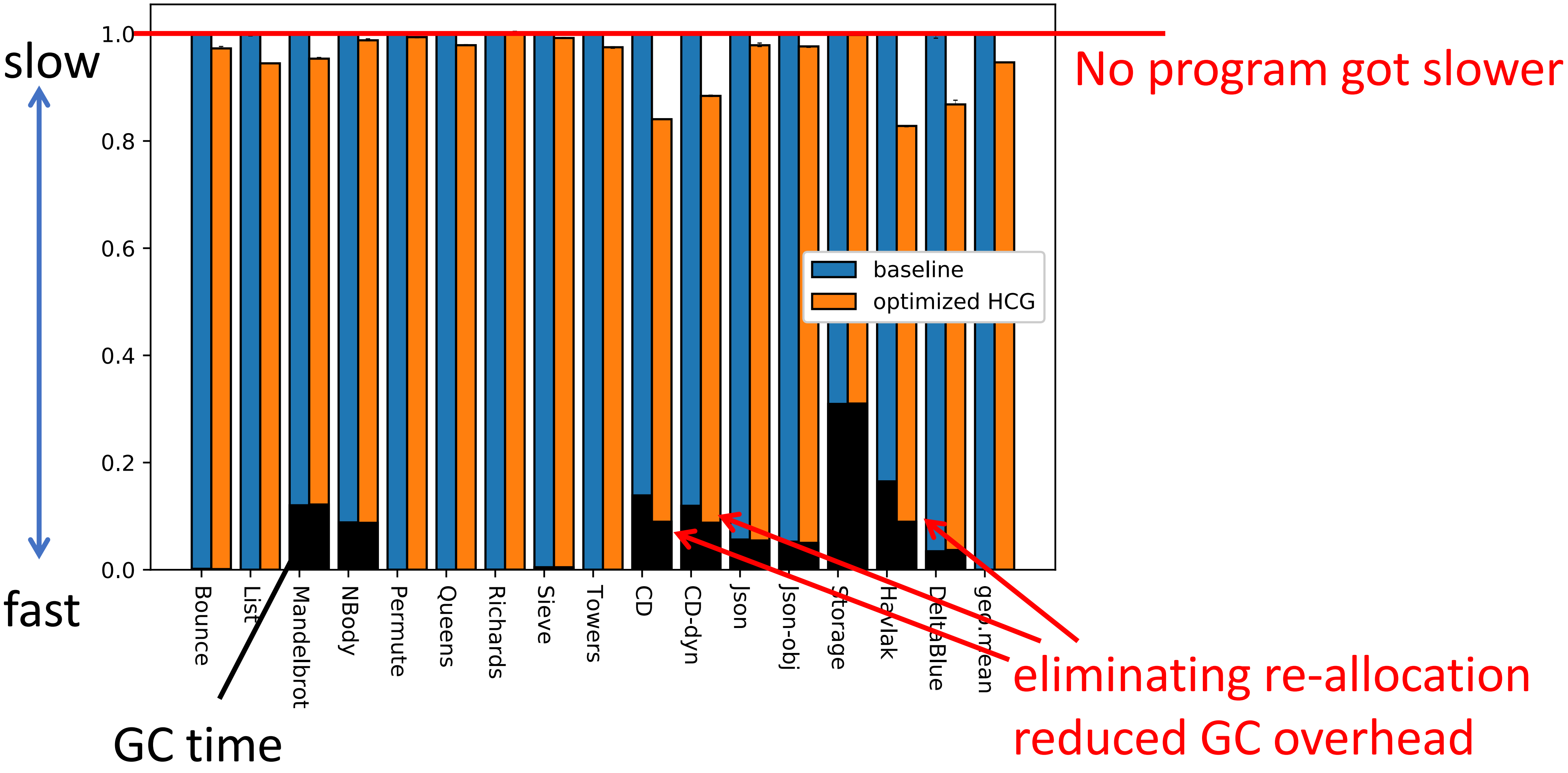
Maximum volume of all objects

including HC-related data



reduced by 17%

Normalised elapsed time

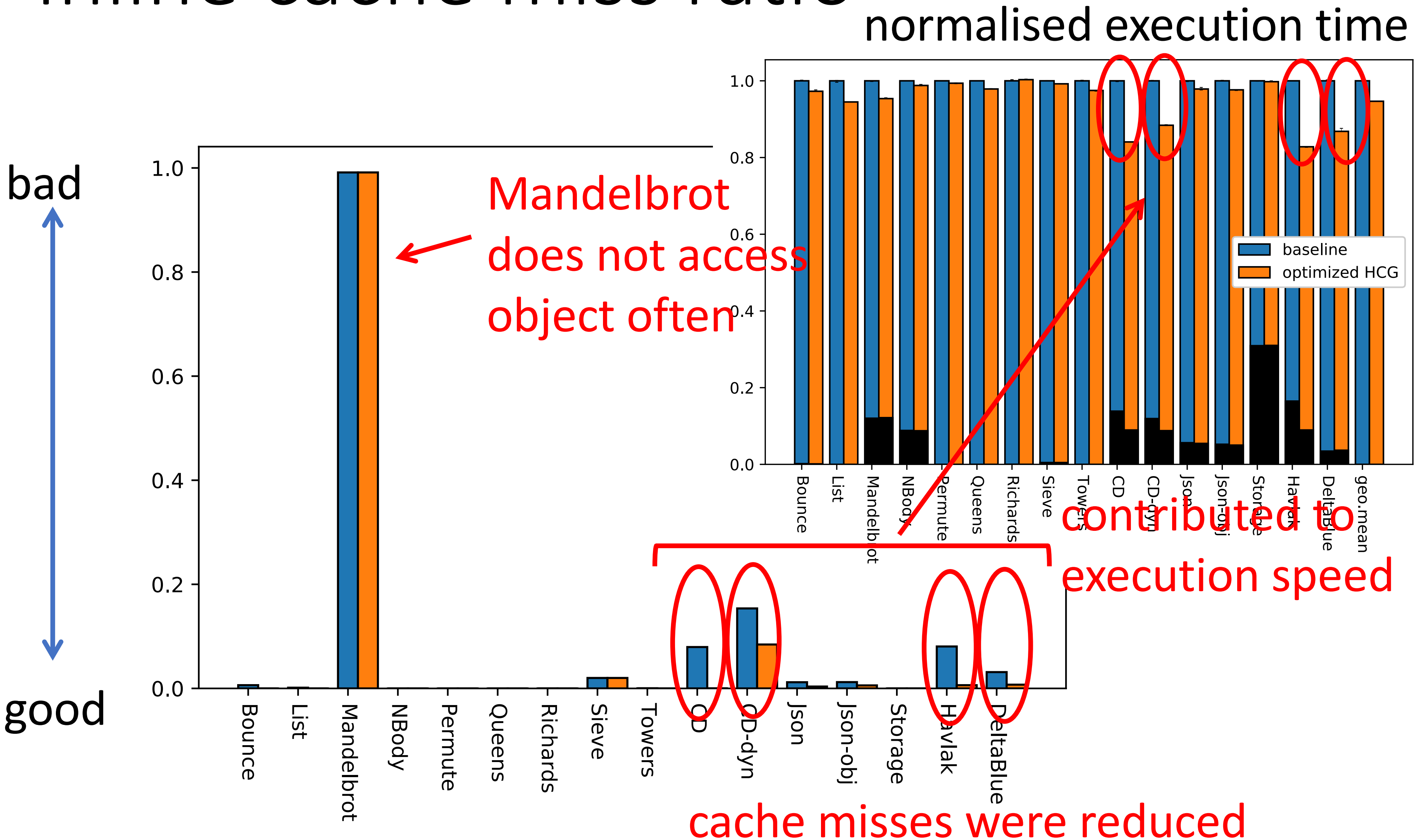


GC time

No program got slower

eliminating re-allocation reduced GC overhead

Inline cache miss ratio



Mandelbrot does not access object often

contributed to execution speed

cache misses were reduced

Conclusion

- We proposed offline optimisation of HCG
    - Move “unlikely” branches
    - Eliminate intermediate HCs
        - preserve popular HCs
- Reduced HC-related data by 61.9% and footprint by 17%
- No program got slower
- Moving branches improved inline cache hit ratio