ActiveContext

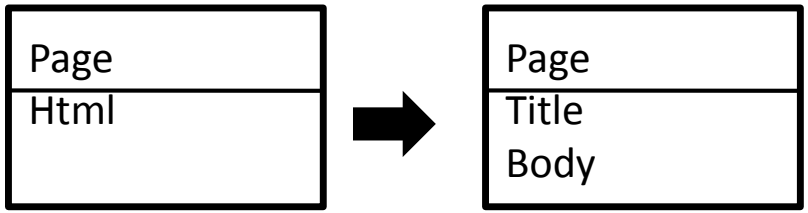
Erwann Wernli

Outline

- Why dyn. software update is challenging (5’)
- Our approach (5’)
- Current stage of the research (5’)

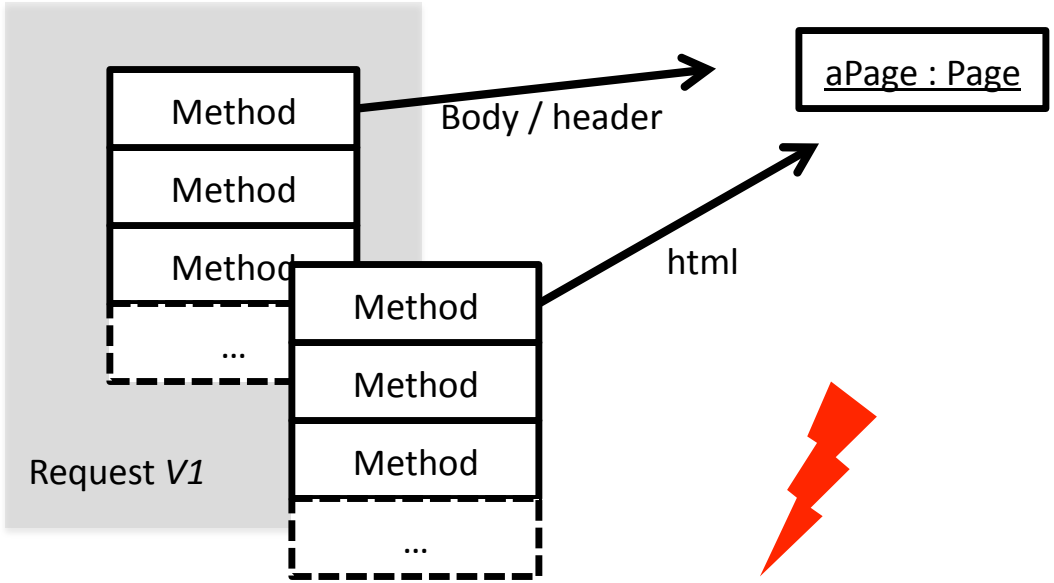
About safety

- Let’s consider the evolution

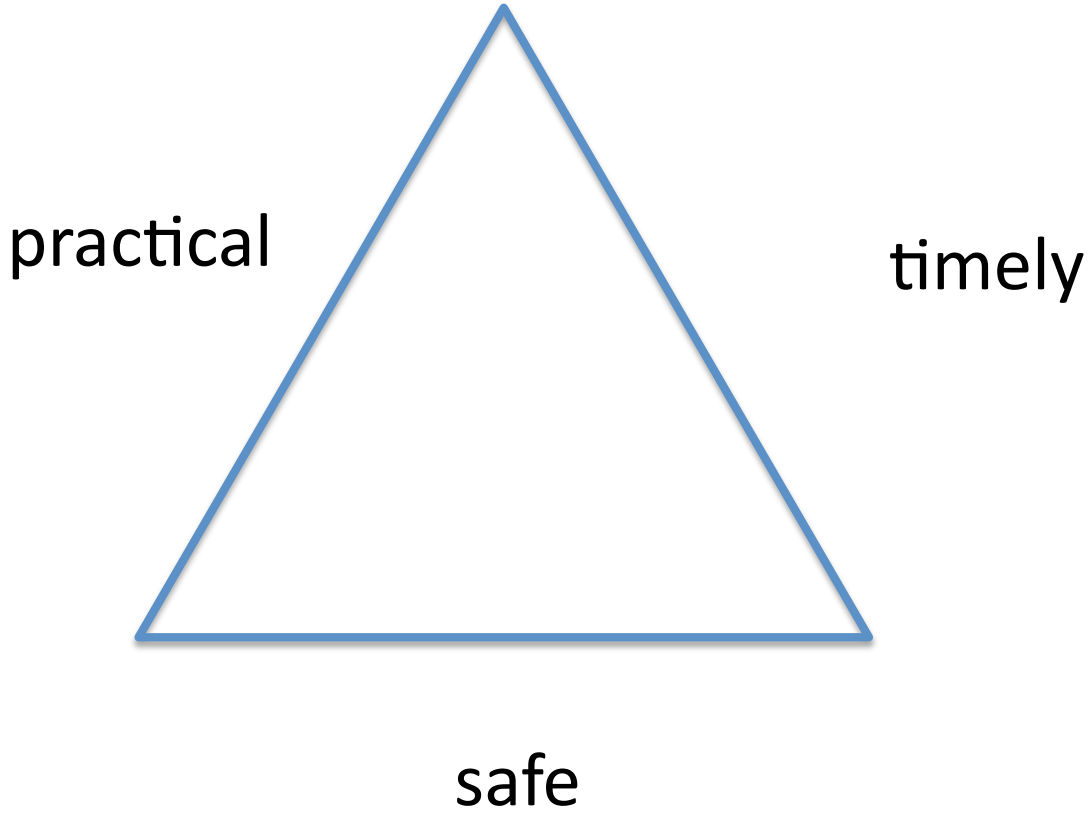


About safety

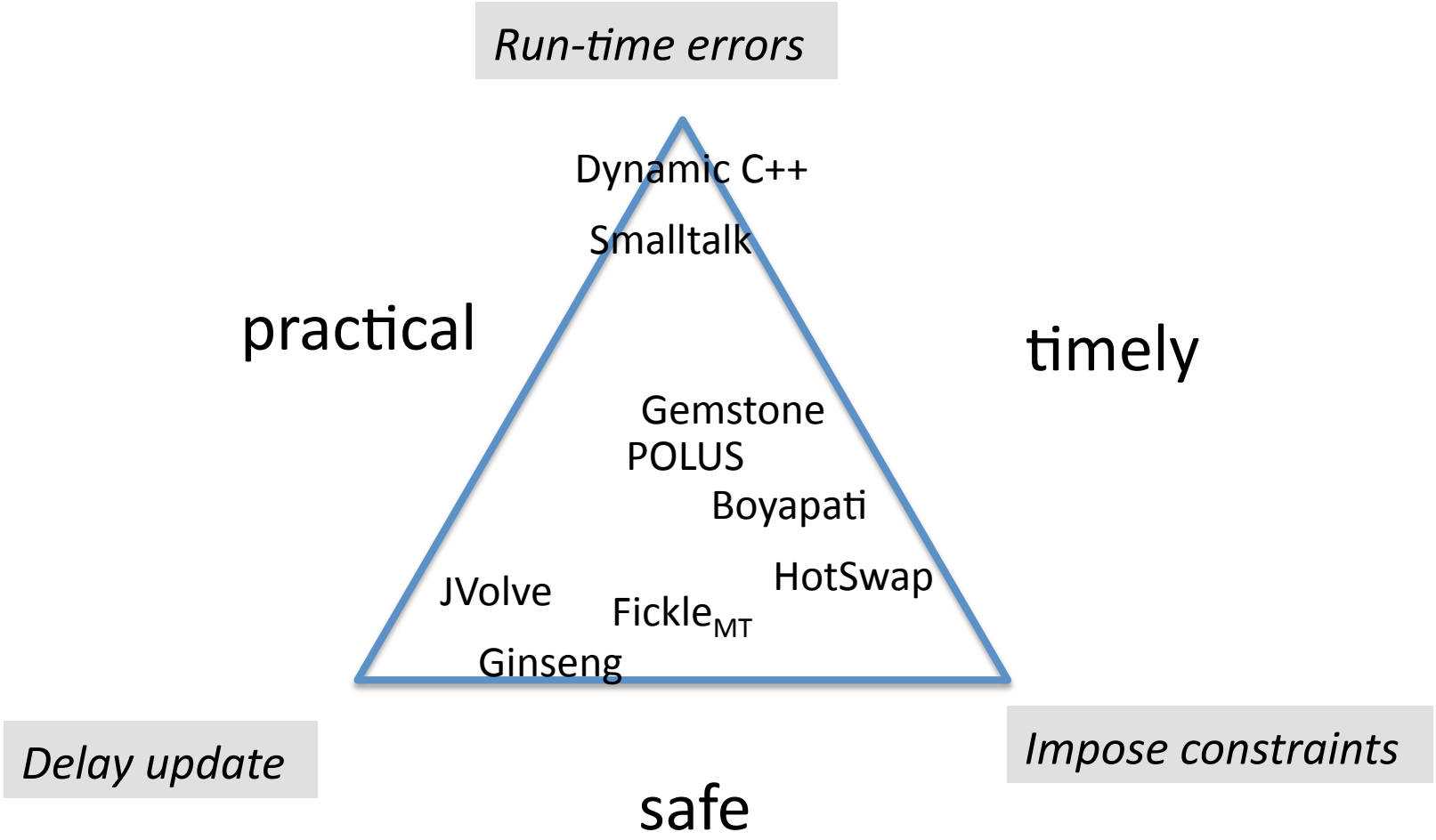
- How can we migrate the state?
- What happens if old running code accesses field `html` ?
- What if an execution mixes old and new code?



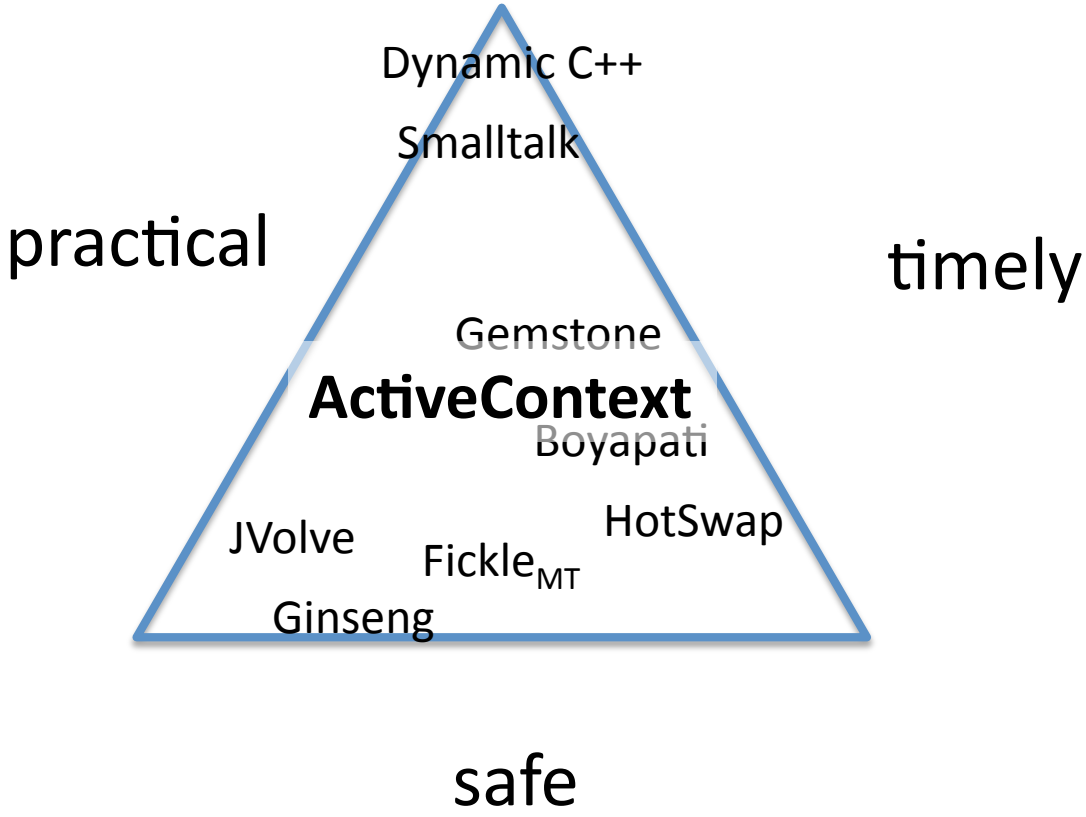
Trade-offs



Trade-offs


Run-time errors
Dynamic C++
Smalltalk
practical
timely
Gemstone
POLUS
Boyapati
HotSwap
JVolve
Fickle$_{MT}$
Ginseng
Delay update
Impose constraints
safe

Trade-offs

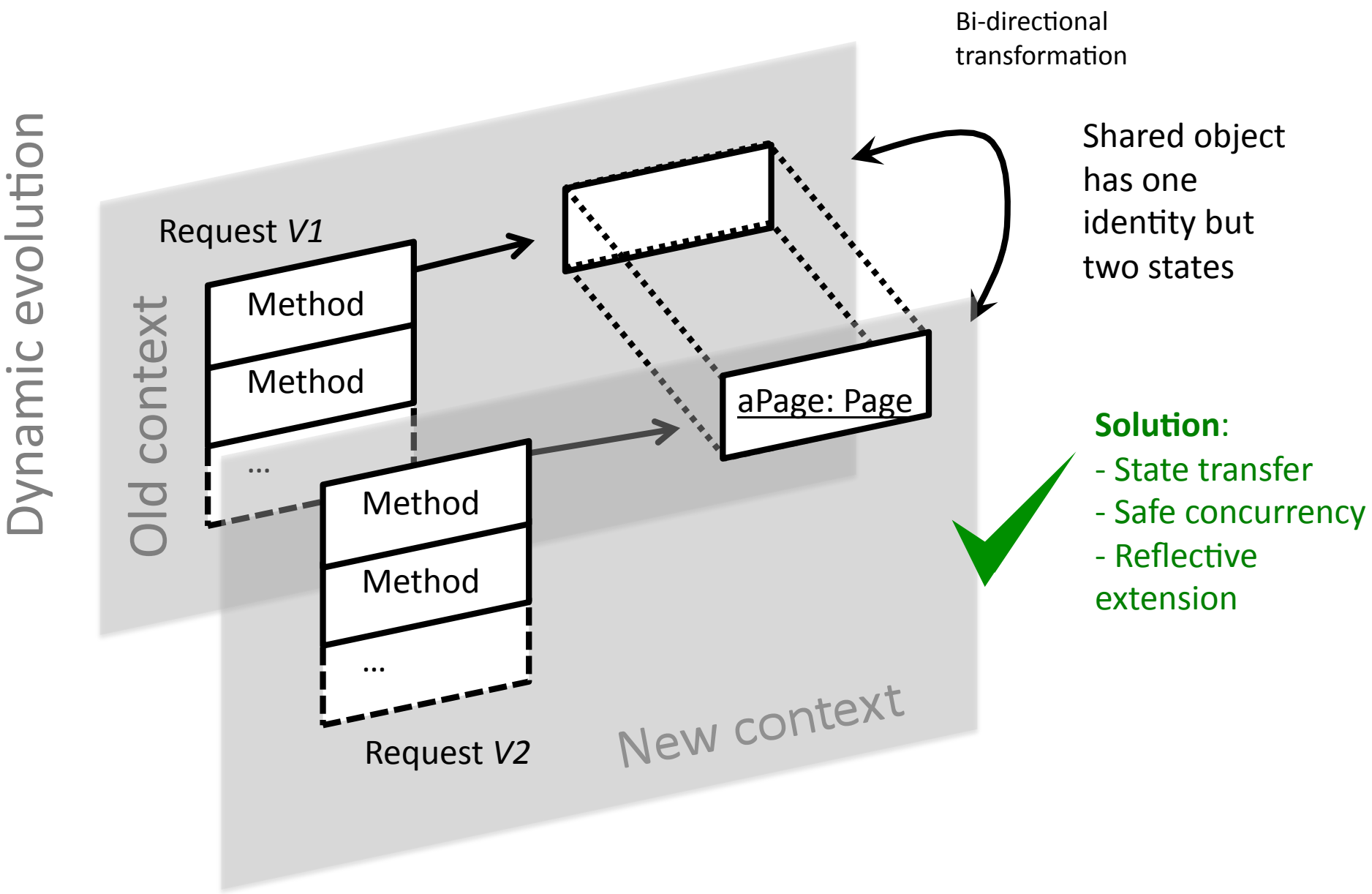

Dynamic C++
Smalltalk
practical
timely
Gemstone
ActiveContext
Boyapati
HotSwap
JVolve
Fickle$_{MT}$
Ginseng
safe

Our approach

a context 

is always version-consistent

```
html := aPage html.
newCtx := NewContext ancesor: oldCtx.
newCtx do: [ aPage header ,
             aPage body ].
```


Dynamic evolution
Old context
Request V1
Method
Method
...
Request V2
Method
Method
...
New context
Bi-directional
transformation
aPage: Page
Shared object
has one
identity but
two states
Solution:
- State transfer
- Safe concurrency
- Reflective
extension

```
1.  transformFromAncestor: id
2.      | cls html body header |
3.      cls := ancestor readClassFor: id.
4.      ( cls = Page ) ifTrue: [
5.          html := ancestor readField: 'html' for: id.
6.          html isNil ifFalse: [
7.              body:= html regex: '<body>(.*)</body>'.
8.              header:= html regex: '<header>(.*)</header>'.
9.          ].
10.         self writeClassFor: id value: Page2.
11.         self writeField: 'body' for: id value: body.
12.         self writeField: 'header' for: id value: header.
13.     ]
14.     ( cls = AnotherClass ) ifTrue: [
15.         ...
16.     ]
17.     ...
```

Design

Strategy	Instantiation	Concurrent read/write	Garbage collection	Expected overhead
Eager	Migrate objects	Sync on write	Garbage when not used	High
Lazy	Does nothing	Sync on dirty read	Force migration prio to GC	Medium

Conclusion

- First-class contexts ensure version consistency
- Safe dynamic software update
- Working implementation
- Future work: enable lazy sync & GC

Design

Contextual first-class classes	Impact
yes	Different versions are the same object Nice with inheritance Class-side variable synchronized
no	Different name per context Not nice with inheritance Class-side variable not synchronized

Validation

- What about daemon threads?
- Can we evole class hierarchies this way?
  (interdependent classes,inheritance)
- Is the run-time overhead acceptable?
  (in space & time)

Safety: no type error, no functional inconsistency

Time: *quickly*, or at least, *eventually* install the update

Practicality: no extra constraint during development