

変化している世界を変化しよう

双方向プログラミングの理論と実践

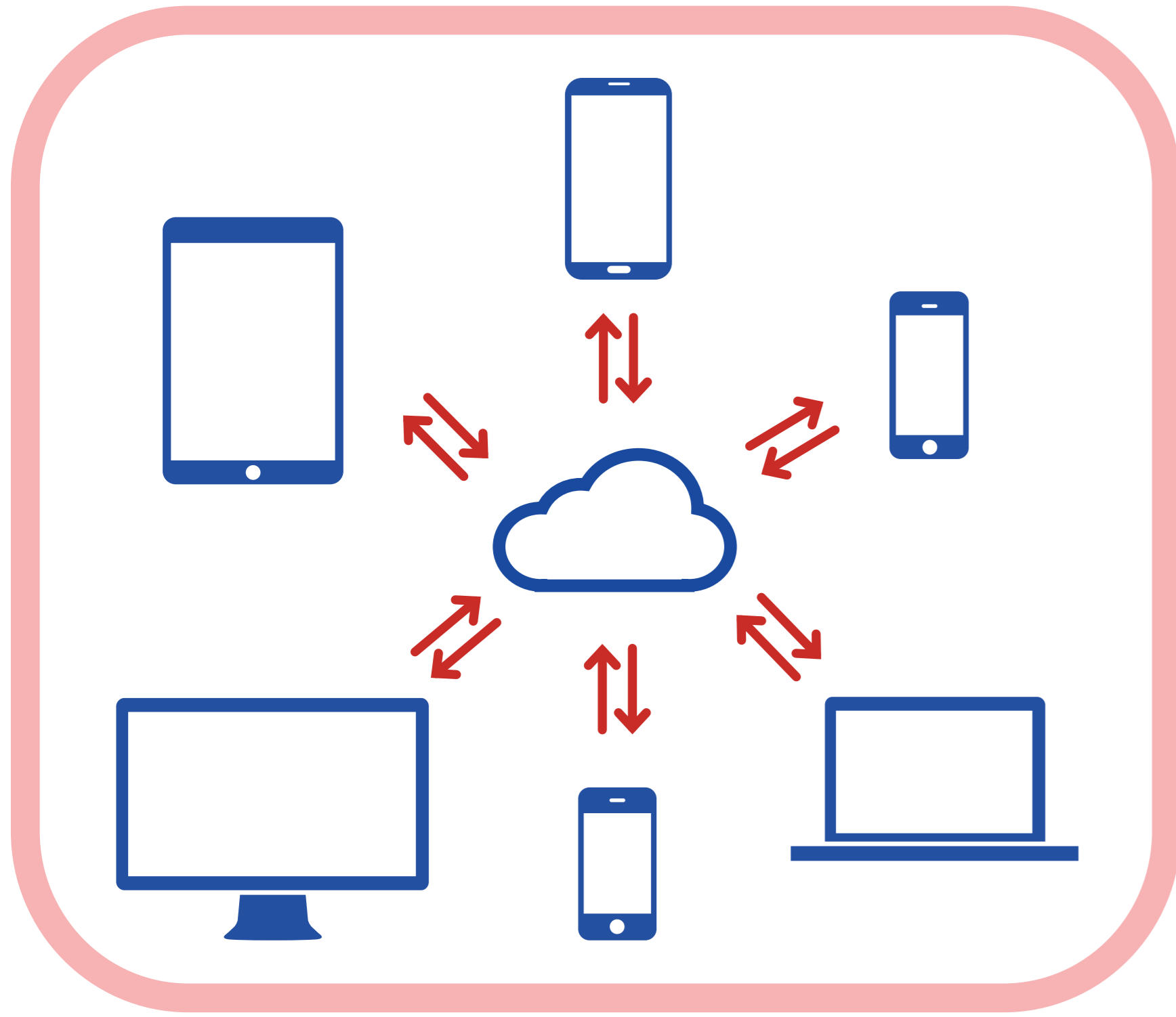
Theory and Practice of Bidirectional Programming

理論と実践

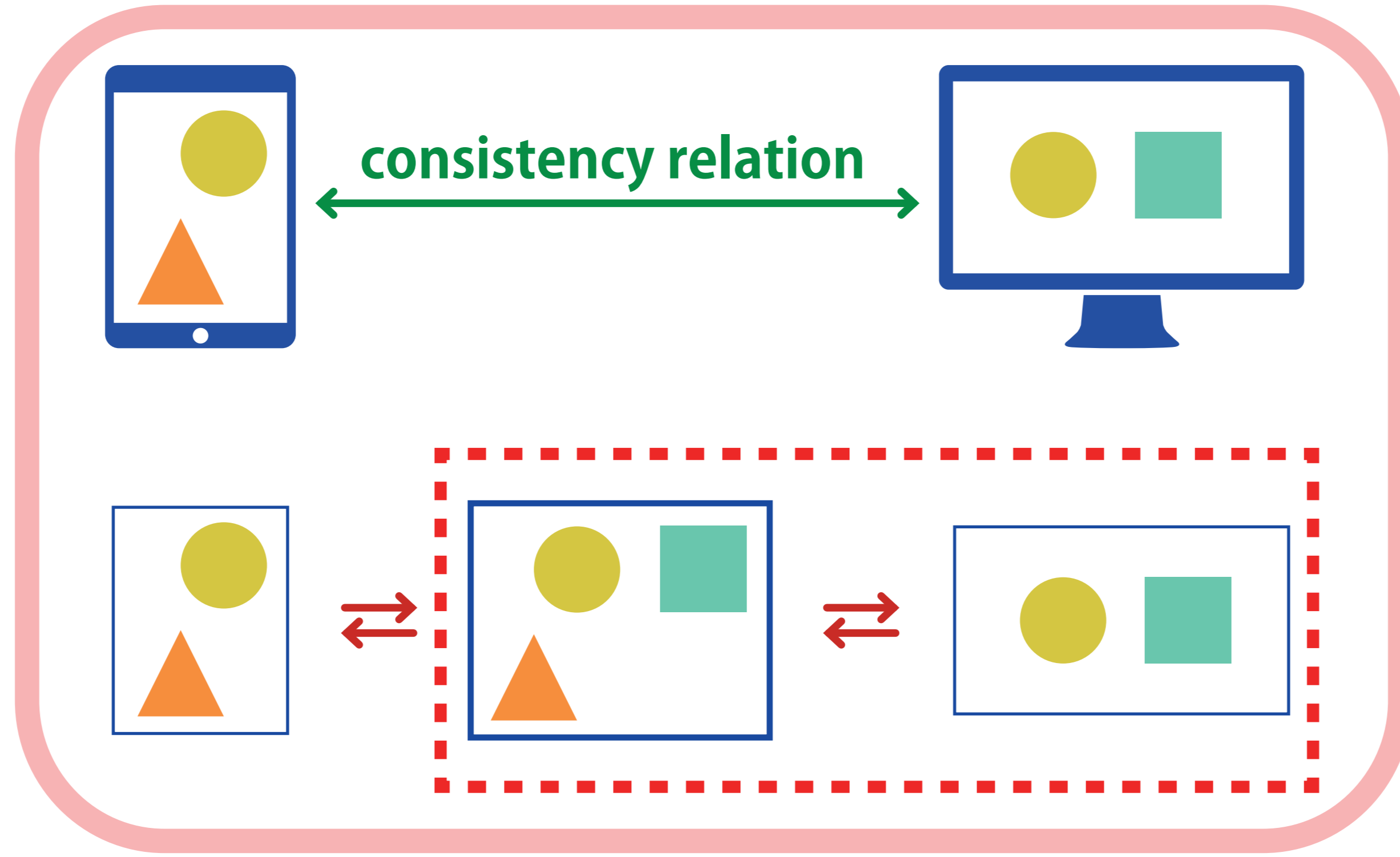
柯 向上 (Hsiang-Shang KO)

朱 子润 (Zirun ZHU)

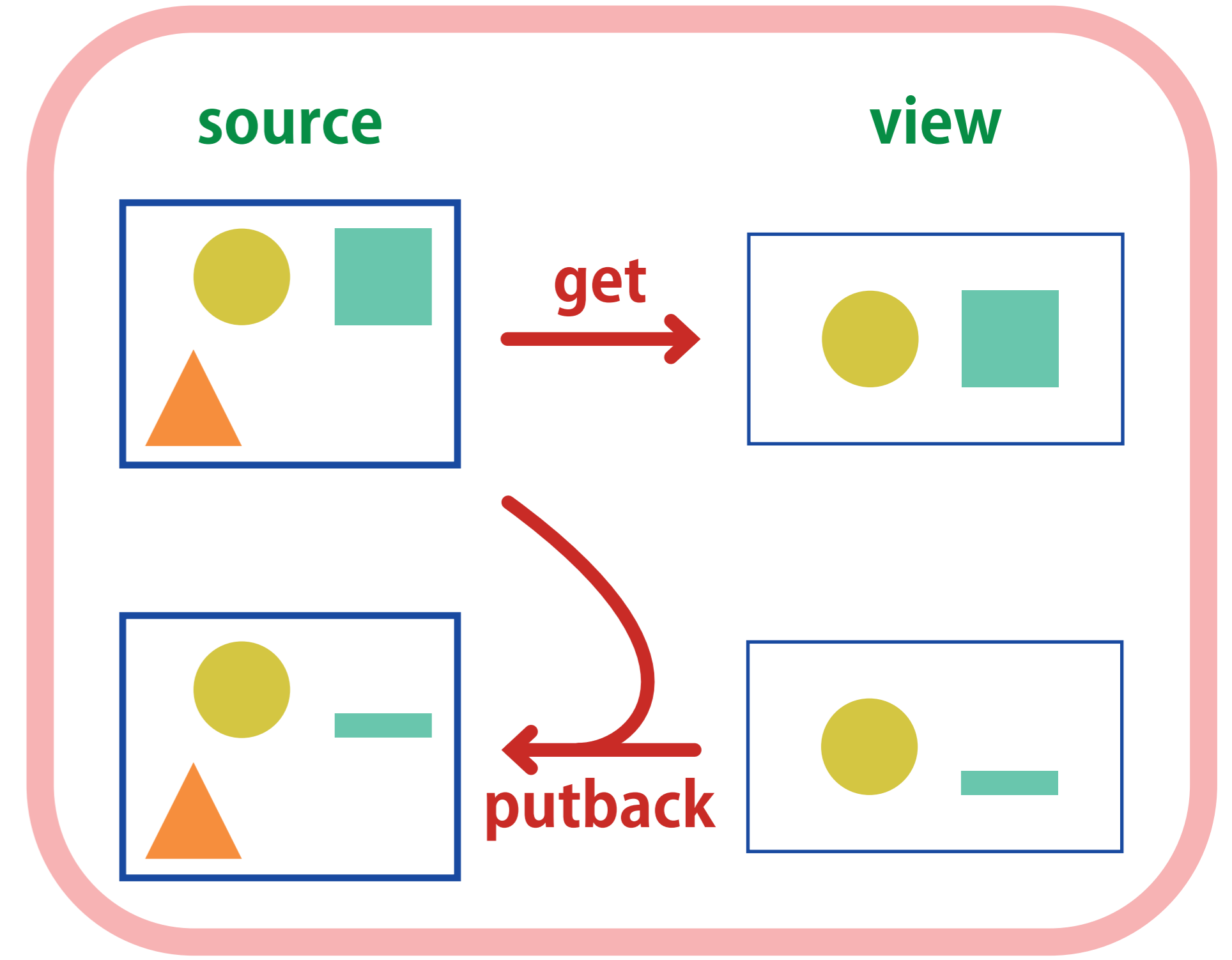
胡 振江 (Zhenjiang HU)



The world needs reliable ways of **synchronisation**.



We can focus on the problem of synchronising two pieces of data, where one side contains more information.



This is studied by the theory of **bidirectional transformations**.

Programming Bidirectional Transformations

We have developed a programming language **BiGUL** , in which every program denotes a well-behaved bidirectional transformation.

case skip Fail Replace
Compose update rearry
rearrs Dep prod

BiGUL consists of a set of small, carefully designed bidirectional components, whose well-behavedness has been **formally verified**, achieving high reliability.

Personal and business calendar

```
calendar:
- event:
  name: group meeting
  start: Today 15:00
  end: Today 17:00
  location: Room 2005
  note: prepare slides
  private: False
- event:
  name: lunch
  start: Today 12:00
  end: Today 13:00
  location: Jimbocho
  note:
  private: True
```



View for secretary

```
calendar:
- event:
  name: group meeting
  start: Today 15:00
  end: Today 17:00
  location: Room 2005
```

Room 2006

```
publicEvents :: BiGUL [Event] [(Name, Location)]
publicEvents = Case
  [ $(normalSV [] and . map private [] [p] [] [])
    [ [] and . map private [] ]
    ==> $(rearrV [] \[] -> () [])$
      skip ()
  , $(adaptiveSV [] not . and . map private [] [p] [] [])
    ==> \es _ -> filter private es
  , $(normal [] \ (e:_) (_:_) -> private e []
    [p] (private -> True):(and . map private -> False) [])
    ==> $(rearrS [] \ (e:es) -> es [])$
      publicEvents
  , $(normal [] \ (e:_) ((n, l):_) -> not (private e) && name e == n []
    [p] (private -> False):_ [])
    ==> $(update [p] (Event n l _):xs [] [p] (n, l):xs []
    [d] n = Replace; l = Replace; xs = publicEvents [])
  , $(adaptive [] \ es ((n, l):_) -> n `elem` map name (filter (not . private) es) [])
    ==> \es ((n, l):_) ->
      let e = fromJust (find (\e -> not (private e) && name e == n) es)
          in e : delete e es
  , $(adaptive [] \ es ((n, l):_) -> not (n `elem` map name (filter (not . private) es)) [])
    ==> \es ((n, l):_) -> Event n l False : es ]
```

The components are then used as building blocks of various bidirectional applications.

H-S Ko, T Zan, and Z Hu. BiGUL: A formally verified core language for putback-based bidirectional programming. Partial Evaluation and Program Manipulation (PEPM), ACM, 2016. DOI: 10.1145/2847538.2847544. Z Zhu, Y Zhang, H-S Ko, P Martins, J Saraiva, and Z Hu. Parsing and reflective printing, bidirectionally. Software Language Engineering (SLE), ACM, 2016. DOI: 10.1145/2997364.2997369.