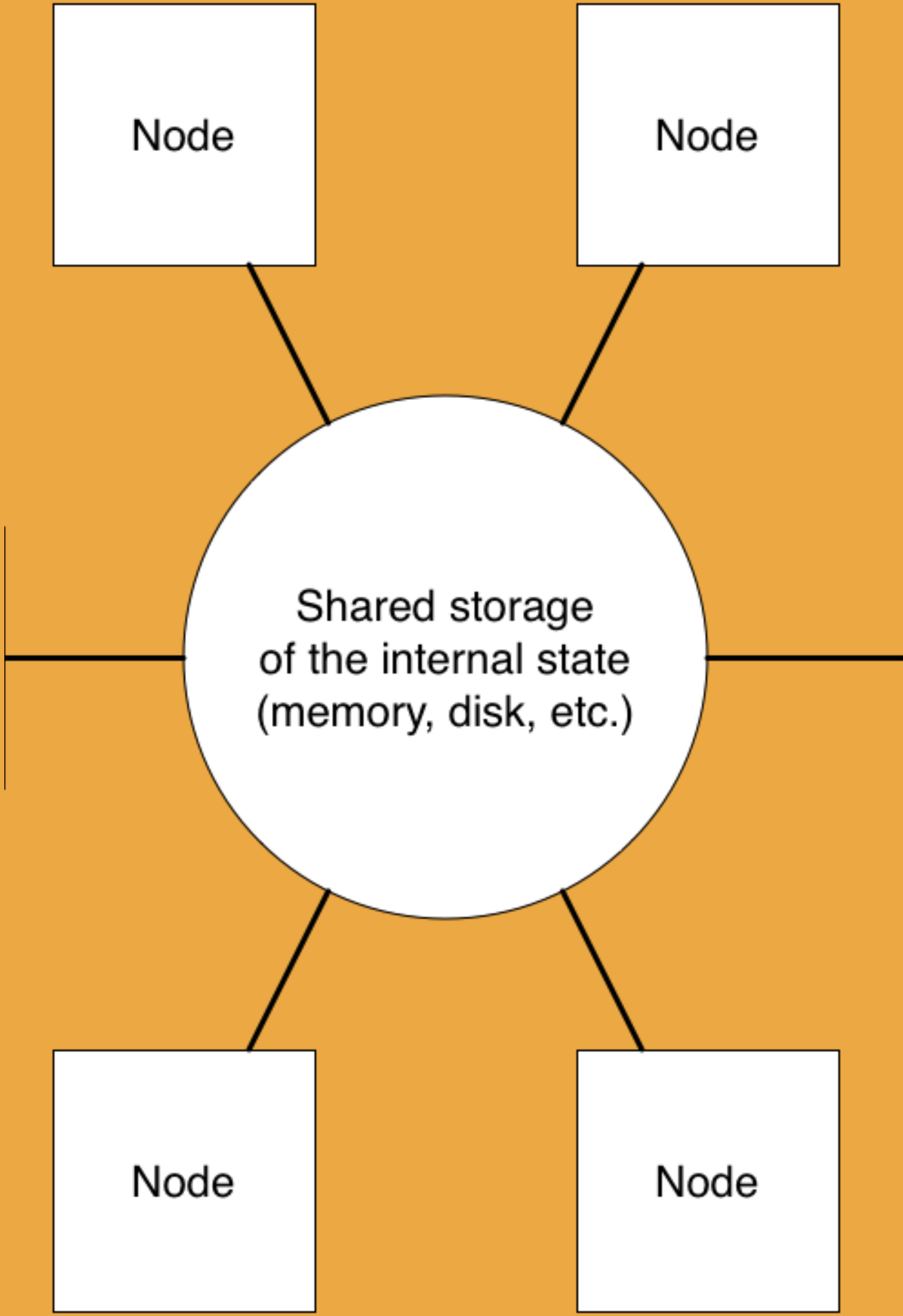
Shared Nothing Secure Programming in Erlang/OTP

Kenji Rikitake

6-JUN-2014
IEICE Tech Report ICSS2014-11
IEICE ICSS #26, Kobe, Japan
@jj1bdx

Traditional memory model: Shared Everything



Internet: a place of sharing

People share almost everything

— Facebook, Twitter, Tumblr

Private group sharing uprising

— WhatsApp, LINE, HipChat

Sharing: a grave source of security problems

Information leakage

- Disclosure of *private* images and videos
- *Security by Obscurity doesn't work*
- Weak access control only by *hidden* URLs, could easily be shared
- Misconfigured access scope
- Dropbox's *referer* header problem
- ... and many others

Shared Everything principle: false assumptions on programming

Programmers assume:

- All resources are readily available from *each and every* computing nodes with *zero latency*
- Bandwidth is infinite
- The network is homogenous
- There is only one administrator

(Quoted from Peter Deutsch's *The Eight Fallacies of Distributed Computing*)

Results: unnecessary coupling of modules and functions

- Unexpected changes of shared memory contents will cause heisenbugs
- Locks and mutual exclusion
- Awareness of consistency is always required such as *thread safeness*: actually many library functions are *thread unsafe!*

History of shared-everything programming

Imperative languages

- From C to JavaScript
- Directly change the internal state
- Internal state is *commonly shared* and accessible between multiple functions and modules
- Use *memory pointers* to minimize the number of copying, inherently suggesting: *share as much as you can*

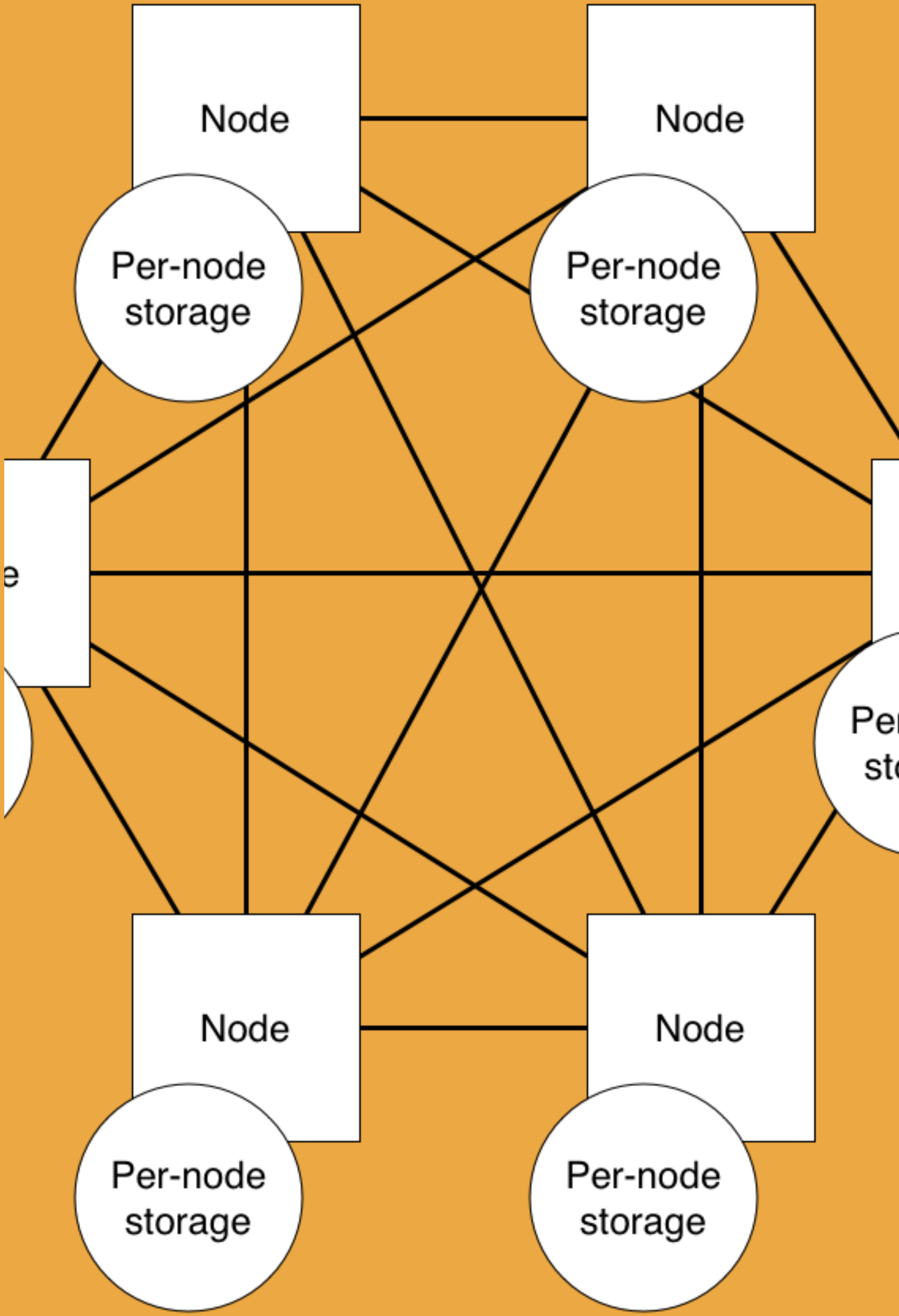
In JavaScript (node.js)

```
// var a = {first: 1, second: 2}
// b = a // only share pointer
{ first: 1, second: 2 }
// a.second = 3
3
// b // element is shared
{ first: 1, second: 3 }
// b == { first: 1, second: 3 }
false // WHY?
```

Cautions on imperative languages

- You cannot share data structures between multiple nodes (they *do not share* the memory addresses)
- Isolating multiple copies of data structures is *not by default*
- Deep comparison of two different data structures *must be externally provided*

Erlang/ OTP's memory model: Shared Nothing (SN)



Shared Nothing principle: the new standard on distributed systems

Shared Nothing architecture system

- No memory sharing among CPUs
- No storage sharing among nodes
- Information exchange should be performed through explicit *message passing* between the nodes

Shared Nothing system's advantages

- Partition tolerance
  - System can be running even when the network is disconnected
  - Availability .vs. consistency
- Isolation of components
  - You *can* make a system running even when a node is broken
  - Amazon Dynamo, Basho Riak

Shared Nothing system's disadvantages

- Totally opposite views of programming from that of the traditional imperative languages required
- *Slow* = needs lots of memory copying
- Sharing cannot be fully eliminated
  - Internal state management is required on each and every level: functions, modules, nodes, multi-node systems

The key issue: choice of default programming modes

Erlang/OTP's programming principles

- Variables are only assigned *once* in the function scope
- No pointer reference: a variable can contain a whole data structure
- *Always deep comparing* two data structures
- Sharing is possible through process dictionaries and message passing, but *not by default*

in Erlang/OTP

```
% A1 = {1,2,3}.
{1,2,3}
% B1 = A1.
{1,2,3} % another copy
% A2 = setelement(3,A1,4).
{1,2,4}
% B1 =:= {1,2,3}.
true
```

Erlang/OTP's Shared Nothing principle enables:

— Efficient garbage collection
— Process isolation
— Referential transparency
— Idempotency

So what and how Shared Nothing contributes to *Security*?

Security = reliability and more

— Privacy
— Resilience
— Immutability
— Confidentiality
— Accountability
— ... but *reliability first*

How Erlang/OTP SN principle will work?

- It's totally opposite from the traditional imperative language programming
- The programmers must *deliberately* share the internal state
- The default mode is *not* sharing
- This will let the programmers *think*

Shared Nothing trends in devops

- Immutable infrastructure - docker
    - Disposable components
    - Replace the whole VM for revision
- The return of static links - golang
- Deploy tools - Chef, Ansible, Puppet
    - Minimize the config parameters

Open questions

- Can programming failures leading to fatal bugs as *gotofail* and *heartbleed* be reduced?
- Are we all ready to accept the inability of Shared Everything paradigm?
- How can research communities contribute to empower the *security first culture*?
- Is Shared Nothing realistic?