



REmote DIctionary Server

#redis

Ezra Zygmuntowicz
twitter: @ezmobius





- Fast, in memory key/value store
- STRING, LIST, SET & ZSET data types
- Persistence via async snapshots or AOF
- Perfect Data Structure/State/Cache Server

Data Structure Server

Data Structure Server

Key:String => Value:String

Data Structure Server

Key:String => Value:String

Key:String => Value:List

Data Structure Server

Key:String => Value:String

Key:String => Value:List

Key:String => Value:Set

Data Structure Server

Key:String => Value:String

Key:String => Value:List

Key:String => Value:Set

Key:String => Value:Zset

Operations on any Type

- Exists, Del, Type
- Keys, Randomkey
- Rename, RenameNX
- Dbsize, Select, Move, Flushdb, Flushall
- TTL, Expire

Operations on STRING's

- Get, Set, GetSet, SetNX
- Mget, Mset, MgetNX, MsetNX
- Incr, Incrby
- Decr, Decrby

Operations on LISTS's

- Atomic Operations:
- Push/Pop
- Index (array indexing)
- Lrange, Ltrim, Llen
- Blpop, Brpop
- RpopLpush, BRpopLpush

Operations on SET's

- Sadd, Srem, Spop, Smove
- Scard, Sismember, Smembers, Srandmember
- Sinter, Sinterstore, Sunion, Sunionstore
- Sdiff, Sdiffstore

Operations on ZSET's

- Zadd, Zrem, Zincrby
- Zrange, Zrevrange
- Zrangebyscore, Zcard
- Zscore, Zremrangebyscore

Seems cool, but what are the use cases?

- Memcached on Steroids
- Tag Clouds, Leaderboards
- Stat collections, circular log buffers
- Share state between processes
- A/B testing
- REDIStribute your load

Example: Tagging

Example: Tagging

SADD article:42 magick

SADD article:42 unicorns

SADD article:42 rainbows

Example: Tagging

SADD article:42 magick
SADD article:42 unicorns
SADD article:42 rainbows

SADD article:12 magick
SADD article:12 bar
SADD article:12 qux
SADD article:12 foo

Example: Tagging

SADD article:42 magick

SADD article:42 unicorns

SADD article:42 rainbows

SADD article:12 magick

SADD article:12 bar

SADD article:12 qux

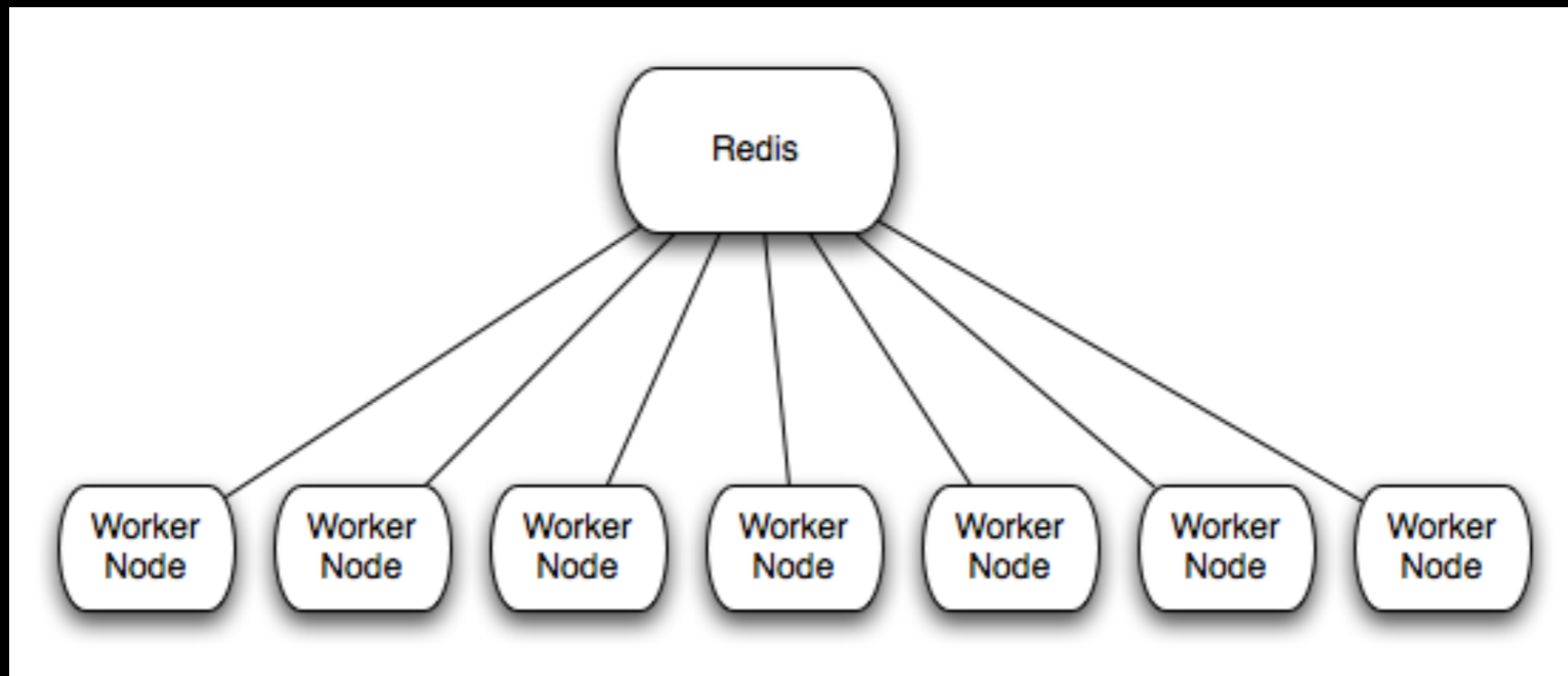
SADD article:12 foo

SINTER article:42 article:12

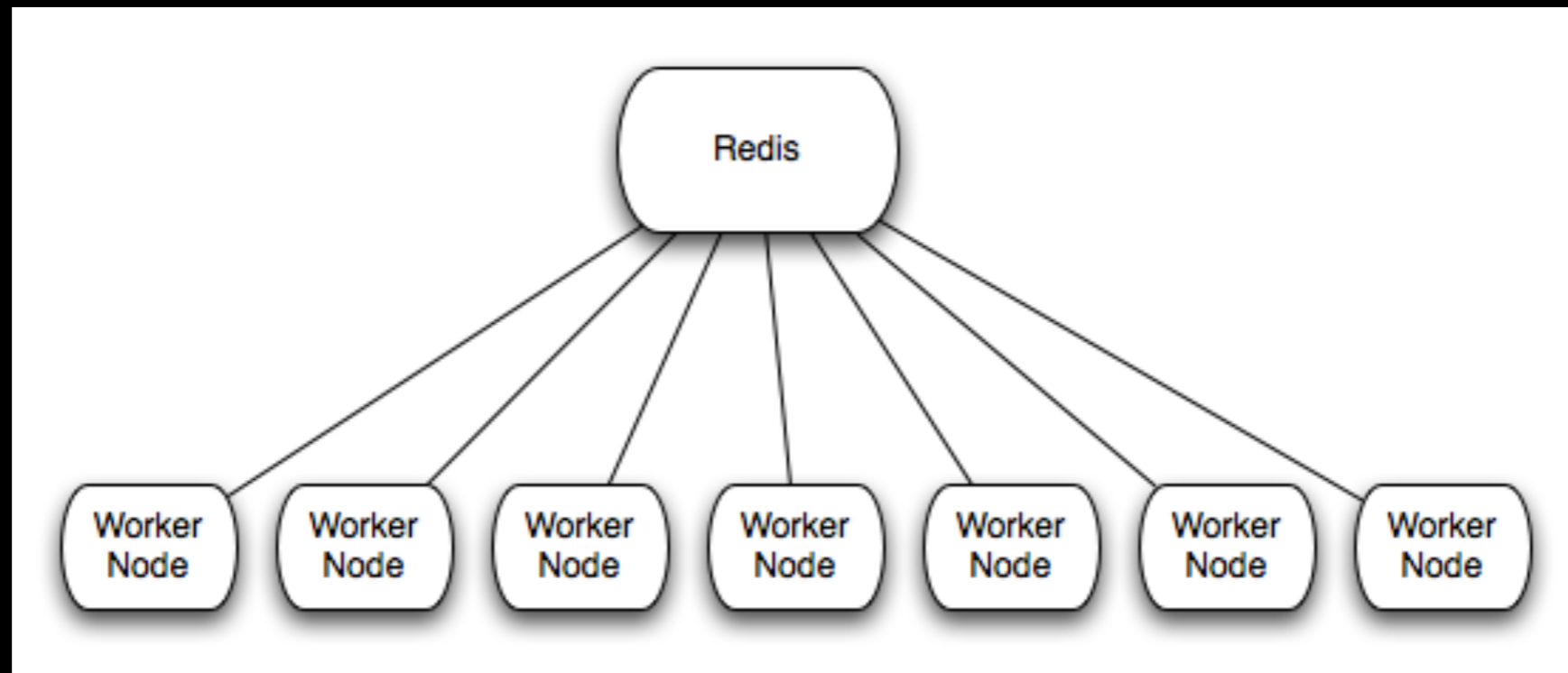
#=> magick

Example: Fair Work Scheduler

Example: Fair Work Scheduler

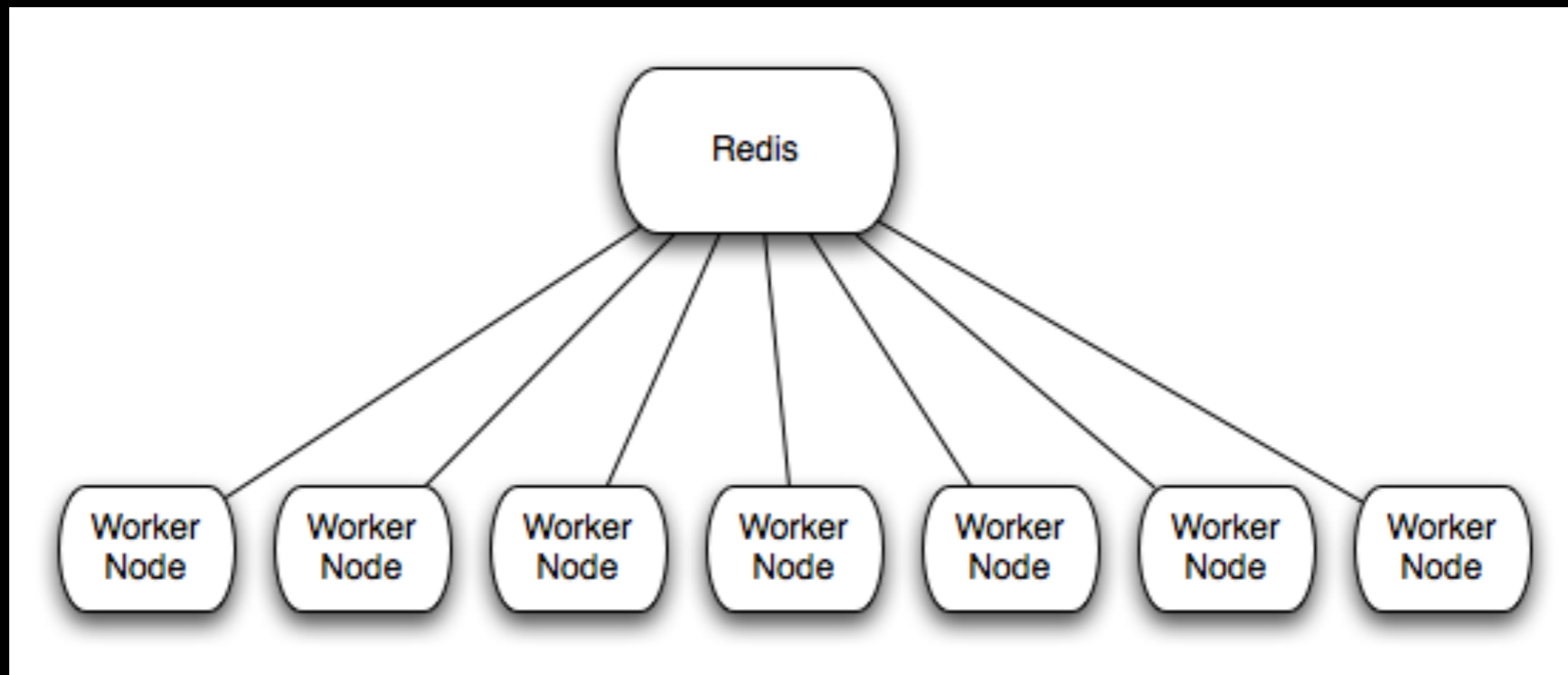


Example: Fair Work Scheduler



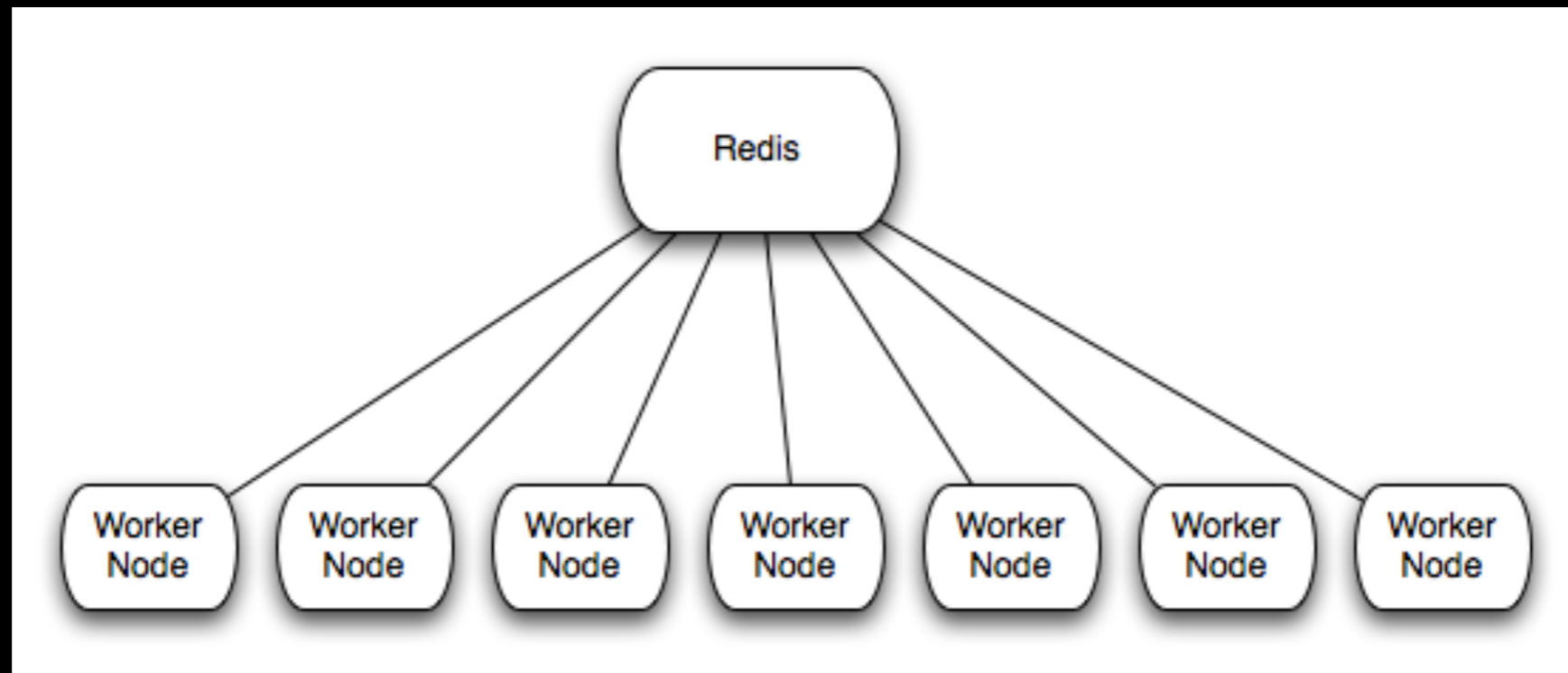
Each worker node periodically issues:
`ZADD worker:nodes 1.5 hostname`
1.5 is the load ave of the host

Example: Fair Work Scheduler



Each worker node listens for work with:
BLPOP hostname 10

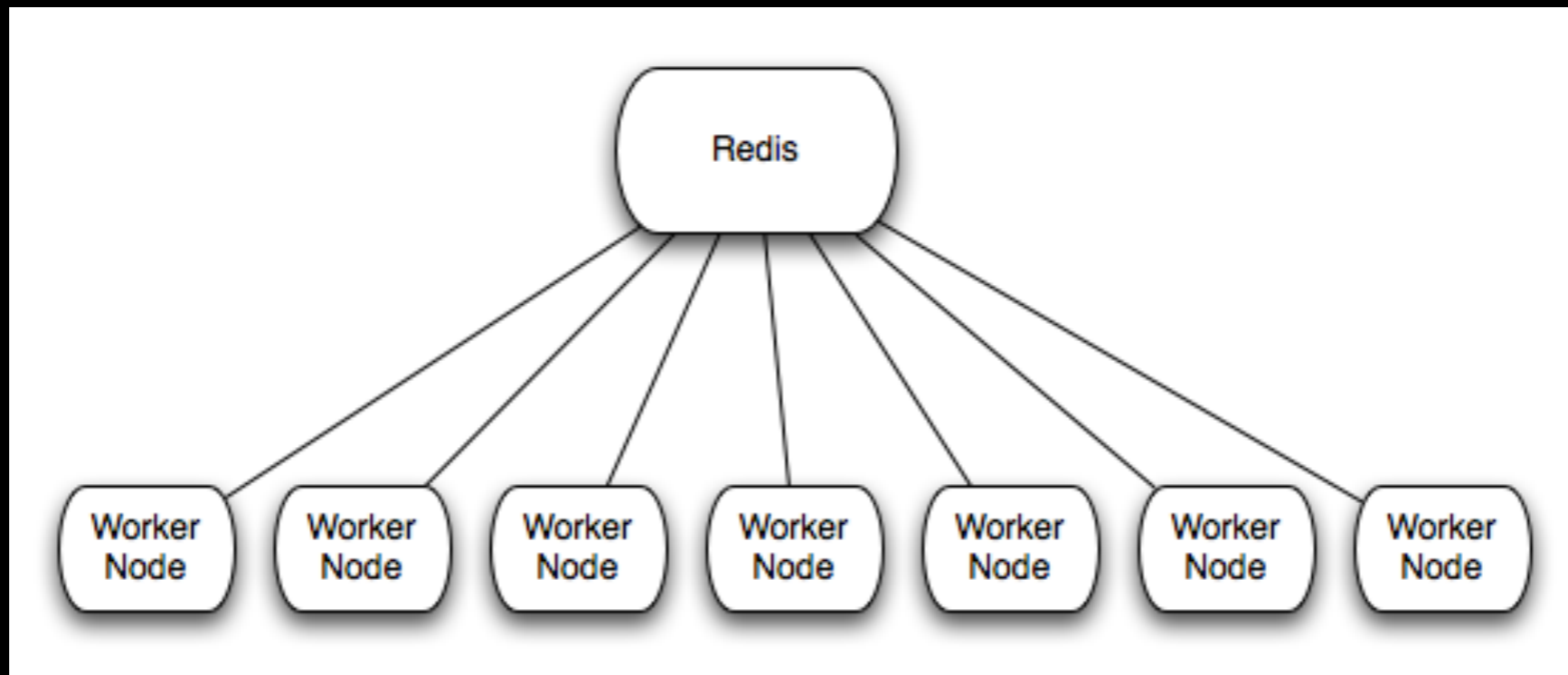
Example: Fair Work Scheduler



When a producer wants to issue a work request:

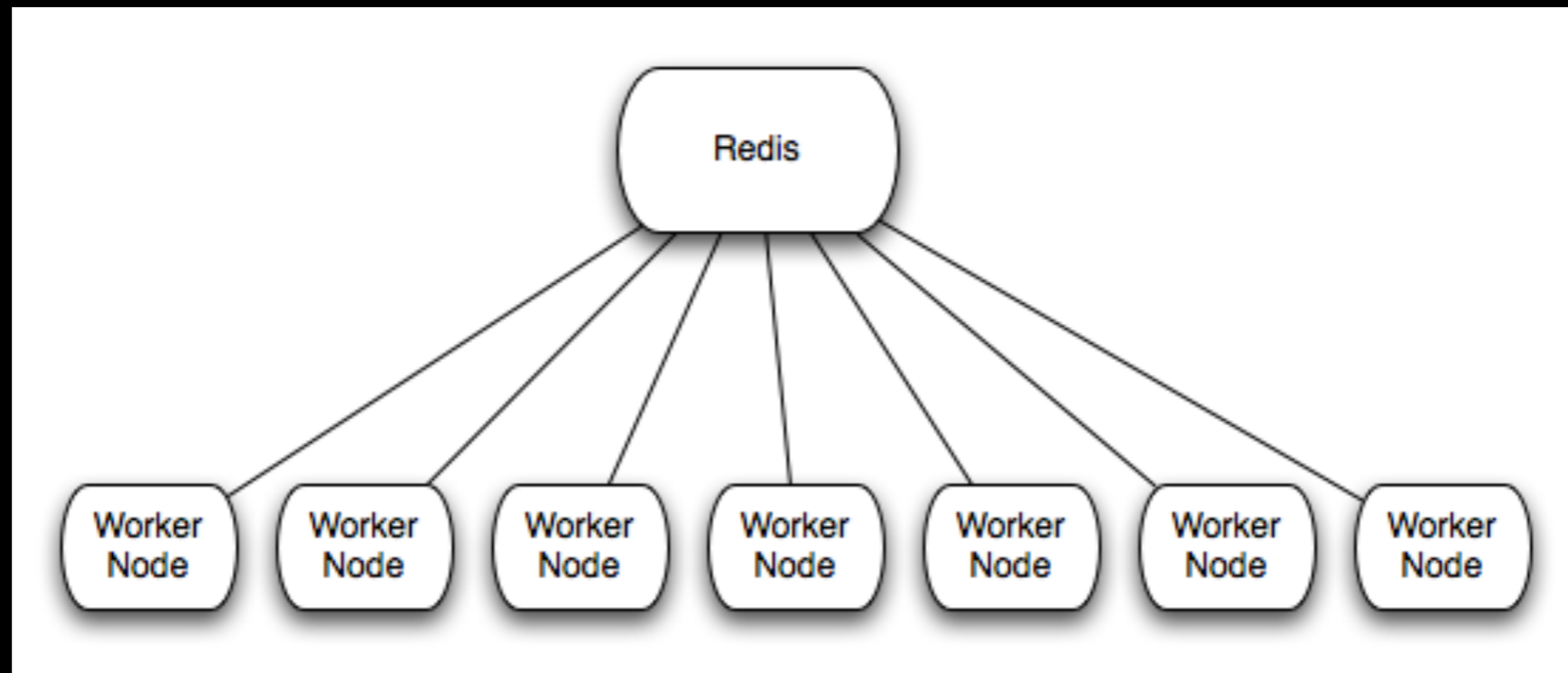
`ZRANGE worker:nodes 0 2`
returns top 3 least loaded nodes

Example: Fair Work Scheduler



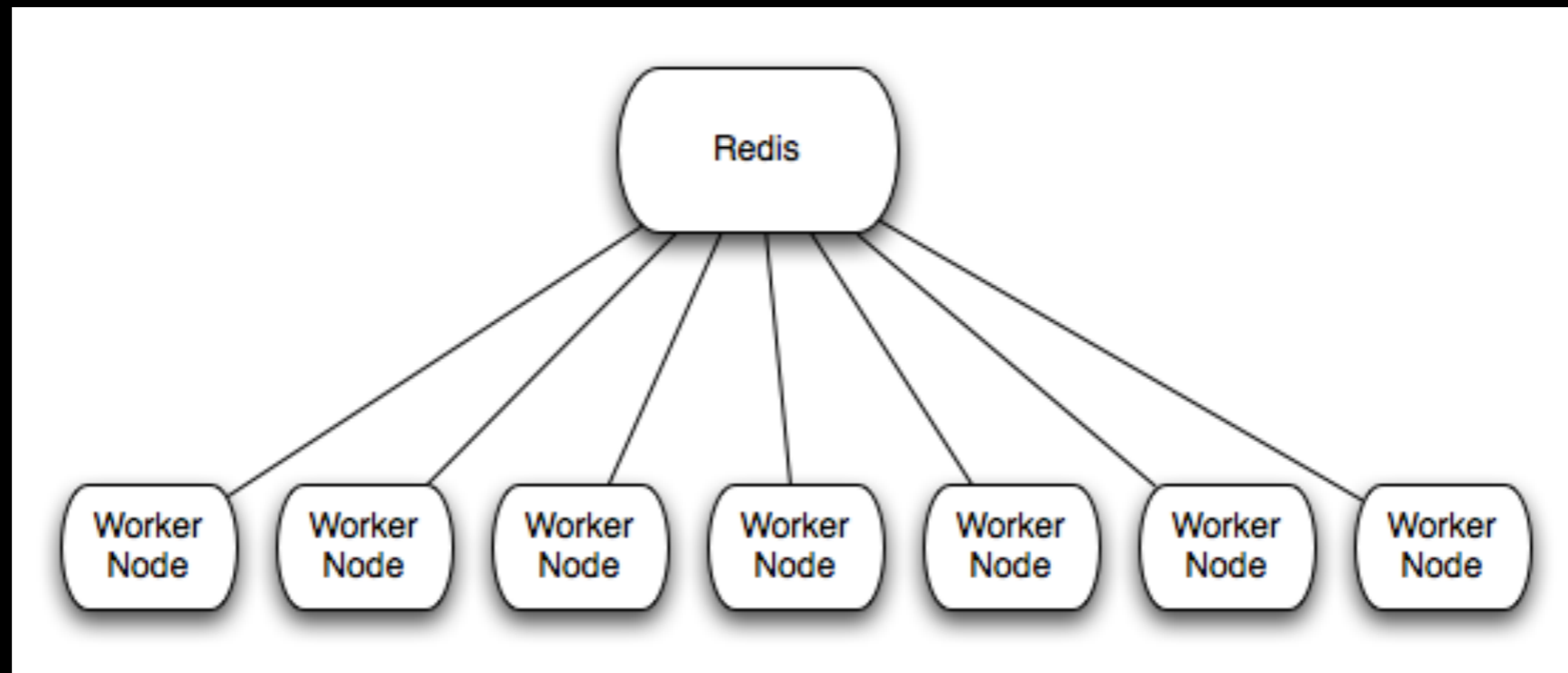
Pick a random node out of the 3 least loaded nodes in order to add jitter so we don't overload hosts

Example: Fair Work Scheduler



Then issue the work request:
LPUSH hostname {"work": {"foo": "bar"}}

Example: Fair Work Scheduler



This setup uses ZSETS and the load average as a score in order to evenly distribute load across a farm of worker instances listening for jobs with BLPOP

Example: AMQP
message de-dupe

Example: AMQP message de-dupe

Route messages through 2 separate rabbitmq brokers for
redundancy

Use redis for message de-dupe by tagging messages with a
guid

Use guid as key into redis

Use SETNX when setting the guid key so first one wins
and second messages gets discarded as already received

Other Ideas

- Distributed Lock Manager for process coordination
- Full Text Inverted Index Lookups
- Caching with extra smarts using lists, sets and atomic ops on said structures
- Share data structures between multiple processes like a blackboard/tuplespace

Speed

- 110k GETS/SETS/second on average linux box
- Mostly faster than memcached for same ops
- Event Driven, can handle thousands of clients with epoll/kqueue support
- Ops happen in memory, persistence is async so everything is very fast

Persistence

- Async Snapshots with configurable triggers
- Append Only File: AOF
- Redis 2.0 Virtual Memory

Replication

- Built in Master -> Slave Async replication
- Create replication chains as needed

Sharding

- Client side sharding (like memcached)
- Consistent ring hashing
- Special case keys “foo{tags}” for operations on keys that must live on the same server in ruby client

Redactor

- Simple Actor Library based around Redis

```
class Foo < RedActor::Actor  
  mailbox :foo  
  mailbox :bar  
  
  def receive_foo(msg)  
    p [:receive_foo, msg]  
    send_msg 'bar', msg + ' world'  
  end  
  
  def receive_bar(msg)  
    p [:receive_bar, msg]  
  end  
  
end  
  
RedActor.run
```

Questions?

```
top - 17:21:27 up 1:20, 4 users, load average: 1.90, 2.23, 1.91
Tasks: 79 total, 1 running, 78 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 71680140k total, 66641604k used, 5038536k free, 400628k buffers
Swap: 0k total, 0k used, 0k free, 816344k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
6609	root	15	0	60.7g	60g	416	S	0	88.7	18:41.56	redis-server /etc/redis.conf
6688	root	16	0	57456	20m	1872	S	0	0.0	1:03.70	irb
6709	root	15	0	58084	18m	1880	S	0	0.0	0:36.62	irb

<http://code.google.com/p/redis/>

<http://github.com/ezmobius/redis-rb>