深入Java多线程和并发编程

ver 1.1

一粟

追風堂

火龙果•整理
uml.org.cn

背景

- 高并发请求的Online应用
  - CPU密集型
  - IO密集型
- 大数据量和计算量的Offline应用
- 培训目的: 建立系统知识，掌握方法

火龙果•整理
uml.org.cn

一些相关知识

- CPU
- 操作系统
  - 中断处理
  - 调度
    - 抢占式
    - 协作式
- 原子操作
  - CAS

火龙果•整理
uml.org.cn

多任务

- 操作系统任务调度
  - 抢占式
  - 协作式
- 多进程
- 多线程
- 协程(coroutine)/纤程(fiber)

火龙果•整理
uml.org.cn

多任务

- 并发 (Concurrency)
  - 一个处理器“同时”处理多个任务
- 并行 (Parallelism)
  - 多个处理器（或多核）“同时”处理多个任务

火龙果•整理
uml.org.cn

多线程的优势

- 可并行处理任务，减少单个任务的等待时间
- 线程较进程或子进程开销小
- 线程间可共享资源
- 多核情况下可充分利用CPU资源

火龙果•整理
uml.org.cn

多线程带来的问题

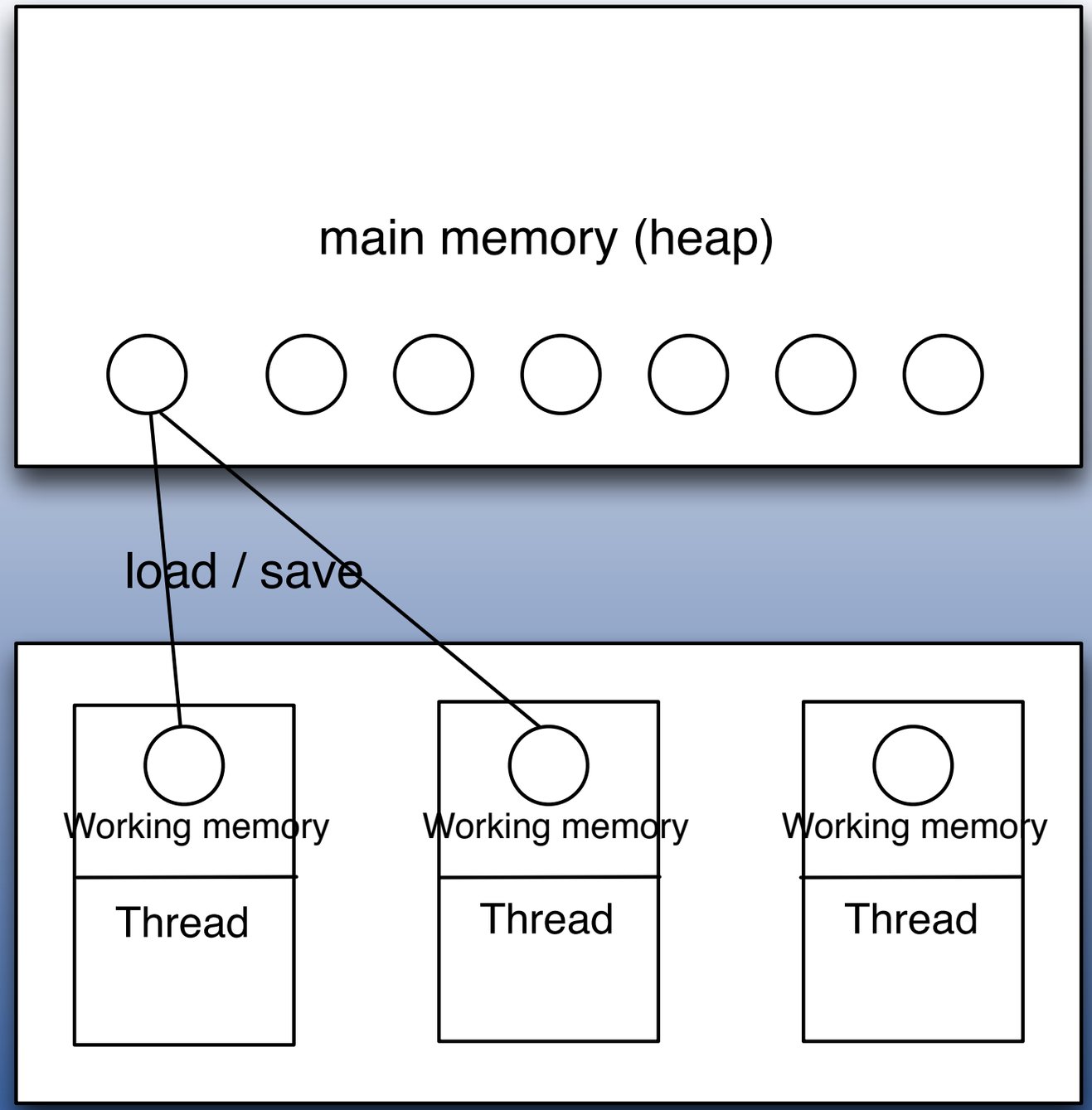
- 访问冲突
- 锁竞争
  - 死锁
  - 锁粒度
- 上下文切换开销
- 同步/内存拷贝开销

火龙果•整理
uml.org.cn

线程安全

- Java 内存模型
  - working memory & main memory (Heap)
- Just-in-time(JIT)编译器优化
- 代码重排 reorder
- 变量可见性

火龙果•整理
uml.org.cn



火龙果•整理
uml.org.cn

happens-before

- 排序原则
  - 对象监视器的解锁发生在等待获取该锁的线程获得锁之前；
  - volitile变量写入操作发生在该变量的读取之前

火龙果•整理
uml.org.cn

```
1  public class NoVisibility {
2      private static boolean ready;
3      private static int number;
4      private static class ReaderThread extends Thread {
5          public void run() {
6              while (!ready) {
7                  Thread.yield();
8              }
9              System.out.println(number);
10         }
11     }
12     public static void main(String[] args) throws Exception {
13         new ReaderThread().start();
14         number = 1;
15         ready = true;
16     }
17 }
```

火龙果•整理
uml.org.cn

Java并发工具

- 同步容器类
- 支持并发的容器类
- 线程池 ExecutorService
- BlockingQueue
- 可重入锁 ReentrantLock，读写锁
- 同步器: CountdownLatch, Semaphore

火龙果•整理
uml.org.cn

同步容器类

- 采用同步锁解决线程安全性能低
- 需要注意多个连续调用仍有线程安全问题
- 迭代过程仍然需要同步

火龙果•整理
uml.org.cn

线程不安全的用法

```
1  public class NotThreadSafe {
2      List<String> list = new ArrayList<String>();
3      Map<String, String> map = new HashMap<String,String>();
4
5      public String getLast() {
6          return list.get(list.size()-1);
7      }
8
9      public void addValue(String key, String value) {
10         for(Map.Entry entry: map.entrySet()) {
11             ...
12         }
13     }
14 }
15
```

火龙果•整理
uml.org.cn

并发容器类

- ConcurrentHashMap
  - 细粒度的锁
  - 弱一致性
- CopyOnWriteArrayList
  - 冗余，不变性

火龙果•整理
uml.org.cn

线程池

```
1 public ThreadPoolExecutor(int corePoolSize,
2                           int maximumPoolSize,
3                           long keepAliveTime,
4                           TimeUnit unit,
5                           BlockingQueue<Runnable> workQueue,
6                           ThreadFactory threadFactory,
7                           RejectedExecutionHandler handler)
```

火龙果•整理
uml.org.cn

线程池配置

- 行为
  - core size不到时候，创建新线程
  - core size到了，任务被放入队列
  - core size到了，并且队列满了，创建新线程直到max size
  - 如果队列满了，max size也达到了，那就开始拒绝任务了...

火龙果•整理
uml.org.cn

线程池配置

- Importance of core size
  - ThreadPoolExecutor changes behavior dramatically around the core size
  - Below core size, threads are always created even if there are idle threads
- Above core size, the preferred behavior shifts to queuing
- Core size should be big enough to accommodate the anticipated average task throughput demand

火龙果•整理
uml.org.cn

线程池配置问题

- 纯CPU计算的应用，线程池应该设多大?
- 一个一半时间都在等待IO的应用，线程池应该设多大?

火龙果•整理
uml.org.cn

线程池配置问题

$$\text{Number of threads} = \frac{\text{Number of Available Cores}}{1 - \text{Blocking Coefficient}}$$

火龙果•整理
uml.org.cn

生产者-消费者问题

- 两个线程
  - 生产者线程：读文件
  - 消费者线程：写文件

火龙果•整理
uml.org.cn

```java
public class Producer extends Thread {
    private BufferedReader reader;
    Buffer buffer;
    public Producer(Buffer buffer) {
        reader = new BufferedReader(new InputStreamReader(System.in));
        this.buffer = buffer;
    }

    void send(String s) {
        buffer.send(s);
    }

    public void run() {
        try {
            while (true) {
                String line = reader.readLine();
                send(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

```java
public class Consumer extends Thread {
    private BufferedWriter writer;
    Buffer buffer;

    public Consumer(Buffer buffer) {
        writer = new BufferedWriter(new OutputStreamWriter(System.out));
        this.buffer = buffer;
    }

    public String receive() {
        return buffer.receive();
    }

    public void run() {
        while (true) {
            String line = receive();
            if (line!=null) {
                try {
                    writer.write(line);
                    writer.flush();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

```

```java
public class Main {
    public static void main(String[] args) {
        Buffer buffer = ...
        Producer producer = new Producer(buffer);
        Consumer consumer = new Consumer(buffer);
        producer.start();
        consumer.start();
    }
}

```

火龙果•整理
uml.org.cn

Polling

```
1  public class PollingBuffer implements Buffer {
2      String line = null;
3
4      @Override
5      public void send(String s) {
6          while (line != null) {
7              pause(1);
8          }
9          line = s;
10     }
11
12     @Override
13     public String receive() {
14         while (line == null) {
15             pause(1);
16         }
17         String s = line;
18         line = null;
19         return s;
20     }
21
22     void pause(int ms) {
23         try {
24             Thread.sleep(ms);
25         } catch (InterruptedException e) {
26             e.printStackTrace();
27         }
28     }
29 }
30
```

火龙果•整理
uml.org.cn

wait/notify/notifyAll

```
1  public class WaitNotifyBuffer implements Buffer {
2      String line;
3      @Override
4      public synchronized void send(String s) {
5          if (line != null) {
6              try {
7                  wait();
8              } catch (InterruptedException e) {
9                  e.printStackTrace();
10             }
11         }
12         line = s;
13         notify();
14     }
15
16     @Override
17     public synchronized String receive() {
18         if (line == null) {
19             try {
20                 wait();
21             } catch (InterruptedException e) {
22                 e.printStackTrace();
23             }
24         }
25         String s = line;
26         line = null;
27         notify();
28         return s;
29     }
30 }
31
```

火龙果•整理
uml.org.cn

Semaphore

```java
public class SemaphoreBuffer implements Buffer {
    String line;
    Semaphore full;
    Semaphore empty;

    public SemaphoreBuffer() {
        full = new Semaphore(1);
        empty = new Semaphore(1);
    }

    @Override
    public void send(String s) {
        try {
            full.acquireUninterruptibly();
            line = s;
        }
        finally {
            empty.release();
        }
    }

    @Override
    public String receive() {
        try {
            empty.acquireUninterruptibly();
            return line;
        }
        finally {
            full.release();
            line = null;
        }
    }
}
```

火龙果•整理
uml.org.cn

BlockingQueue

```
1  public class BlockingQueueBuffer implements Buffer {
2      BlockingQueue<String> queue;
3
4      public BlockingQueueBuffer (int capacity) {
5          queue = new ArrayBlockingQueue<String>(capacity);
6      }
7
8      @Override
9      public void send(String s) {
10         try {
11             queue.put(s);
12         } catch (InterruptedException e) {
13             e.printStackTrace();
14         }
15     }
16
17     @Override
18     public String receive() {
19         try {
20             return queue.take();
21         } catch (InterruptedException e) {
22             e.printStackTrace();
23         }
24         return null;
25     }
26 }
```

火龙果•整理
uml.org.cn

多读少写的并发问题

1. 完全同步

```
1  class Synchronized {
2      private final List<String> list = new ArrayList<String>();
3
4      public synchronized void iterateOnList() {
5          for(String s: list) {
6              //do something with s
7          }
8      }
9
10     public synchronized void add(String value) {
11         list.add(value);
12     }
13 }
```

火龙果•整理
uml.org.cn

多读少写的并发问题

2. 读写锁

火龙果•整理
uml.org.cn

```
1  class UsingReadWriteLock {
2      private final List<String> list = new ArrayList<String>();
3      private final ReadWriteLock lock = new ReentrantReadWriteLock();
4
5      public void iterateOnList() {
6          lock.readLock().lock();
7          try {
8              for(String s: list) {
9                  //do something with s
10             }
11         } finally {
12             lock.readLock().unlock();
13         }
14     }
15
16     public synchronized void add(String value) {
17         lock.writeLock().lock();
18         try {
19             list.add(value);
20         } finally {
21             lock.writeLock().unlock();
22         }
23     }
24 }
25
```

火龙果•整理
uml.org.cn

多读少写的并发问题

3. Copy On Write

```
1  class CopyOnWrite {
2      private volatile List<String> list = new ArrayList<String>();
3
4      public void iterateOnList() {
5          for(String s: list) {
6              //do something with s
7          }
8      }
9
10     public synchronized void add(String value) {
11         List<String> copy = new ArrayList<String>(list);
12         copy.add(value);
13         list = copy;
14     }
15 }
```

火龙果•整理
uml.org.cn

原子性

```
1 public class CounterServlet implements Servlet {
2     private int count = 0;
3
4     public int getCount() { return count; }
5
6     public void service(ServletRequest req,
                          ServletResponse res) {
7         ++count;
8     }
9 }
10
```

火龙果•整理
uml.org.cn

原子性

```
1 public class CounterServlet implements Servlet {
2     private int count = 0;
3
4     public synchronized int getCount() { return count; }
5
6     public synchronized void service(ServletRequest req,
                                       ServletResponse res) {
7         ++count;
8     }
9 }
10
```

火龙果•整理
uml.org.cn

原子性

```
1 public class CounterServlet implements Servlet {
2     private volatile int count = 0;
3
4     public int getCount() { return count; }
5
6     public synchronized void service(ServletRequest req,
                                       ServletResponse res) {
7         ++count;
8     }
9 }
10
```

火龙果•整理
uml.org.cn

案例

```
1 public class CounterServlet implements Servlet {
2     private AtomicInteger count = new AtomicInteger(0);
3
4     public int getCount() { return count.get(); }
5
6     public void service(ServletRequest req,
                          ServletResponse res) {
7         count.incrementAndGet();
8     }
9 }
10
```

火龙果•整理
uml.org.cn

代码找错

```
1  //Bad code
2  public class Poll {
3      private final int MAX = 100;
4      private long count;
5
6      public Poll() {
7          new Thread(new Runnable() {
8              public void run() { add(); }
9          }).start();
10     }
11
12     public long getCountAvailable() { return count; }
13
14     public boolean take(int unit) {
15         if (unit > 0 && count > unit) {
16             count -= unit;
17             return true;
18         }
19         return false;
20     }
21
22     public void add() {
23         while(true) {
24             if (count < MAX) {
25                 count++;
26             }
27             try { Thread.sleep(1000l);} catch (InterruptedException e) {};
28         }
29     }
30 }
31
```

火龙果•整理
uml.org.cn

```java
public class Poll {
    private final int MAX = 100;
    private final AtomicLong count = new AtomicLong(0);
    private final Timer timer = new Timer(true);

    private Poll() {}

    public void init() {
        TimerTask task = new TimerTask() {
            @Override
            public void run() {
                if (count.get() < MAX) {
                    count.incrementAndGet();
                }
            }
        };
        timer.scheduleAtFixedRate(task,0,1000);
    }

    public static final Poll create() {
        Poll poll = new Poll();
        poll.init();
        return poll;
    }

    public long getCountAvailable() {
        return count.get();
    }

    public boolean take(int unit) {
        long current = count.get();
        if (unit > 0 && current > unit) {
            return count.compareAndSet(current, current-unit);
        }
        return false;
    }
}
```

火龙果•整理
uml.org.cn

另一些并发解决方法

- 不可改变对象 Immutable Object
- 无状态的函数式编程
- Java 7 Fork/Join Framework
  - work-stealing algorithm
- Clojure: STM
- Scala/Erlang: actor
- Kilim: coroutine

火龙果•整理
uml.org.cn

Coroutine

- 像线程一样拥有自己栈和局部变量
- 轻量：创建和切换开销小
- 同一时刻只能运行一个协程
- 支持的语言：Lua，Ruby，Python，Go

火龙果•整理
uml.org.cn

Lua Coroutine 例子

```
function receive ()
    local status, value = coroutine.resume(producer)
    return value
end

function send (x)
    coroutine.yield(x)
end

producer = coroutine.create( function ()
    while true do
        local x = io.read() -- produce new value
        send(x)
    end
end)
```

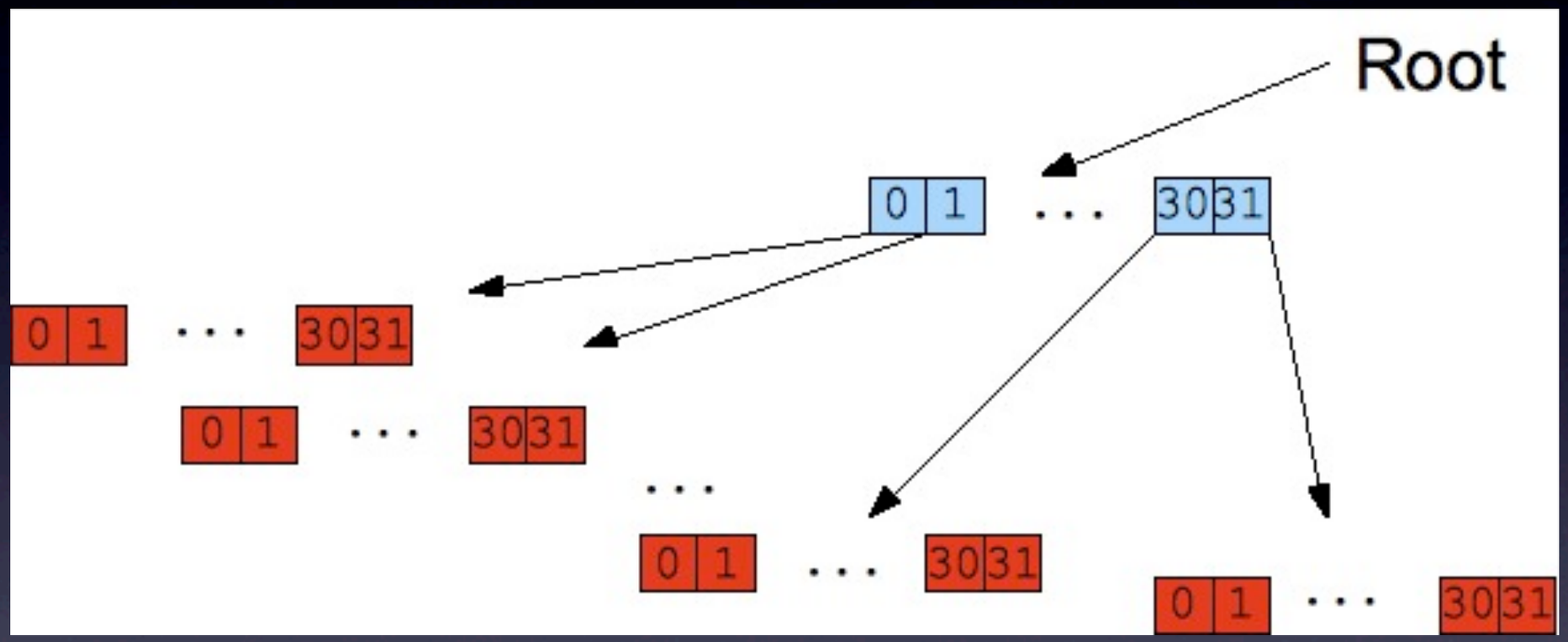
火龙果•整理
uml.org.cn

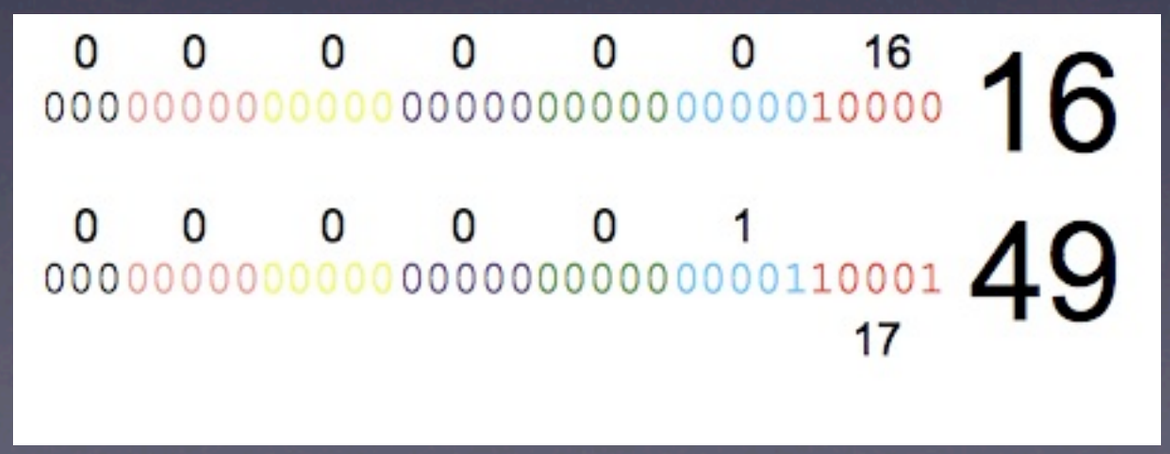
Clojure STM

- Software Transactional Memory
- 类似数据库事务处理（ACID）
- Multiversion Concurrency Control (MVCC)

火龙果•整理
uml.org.cn

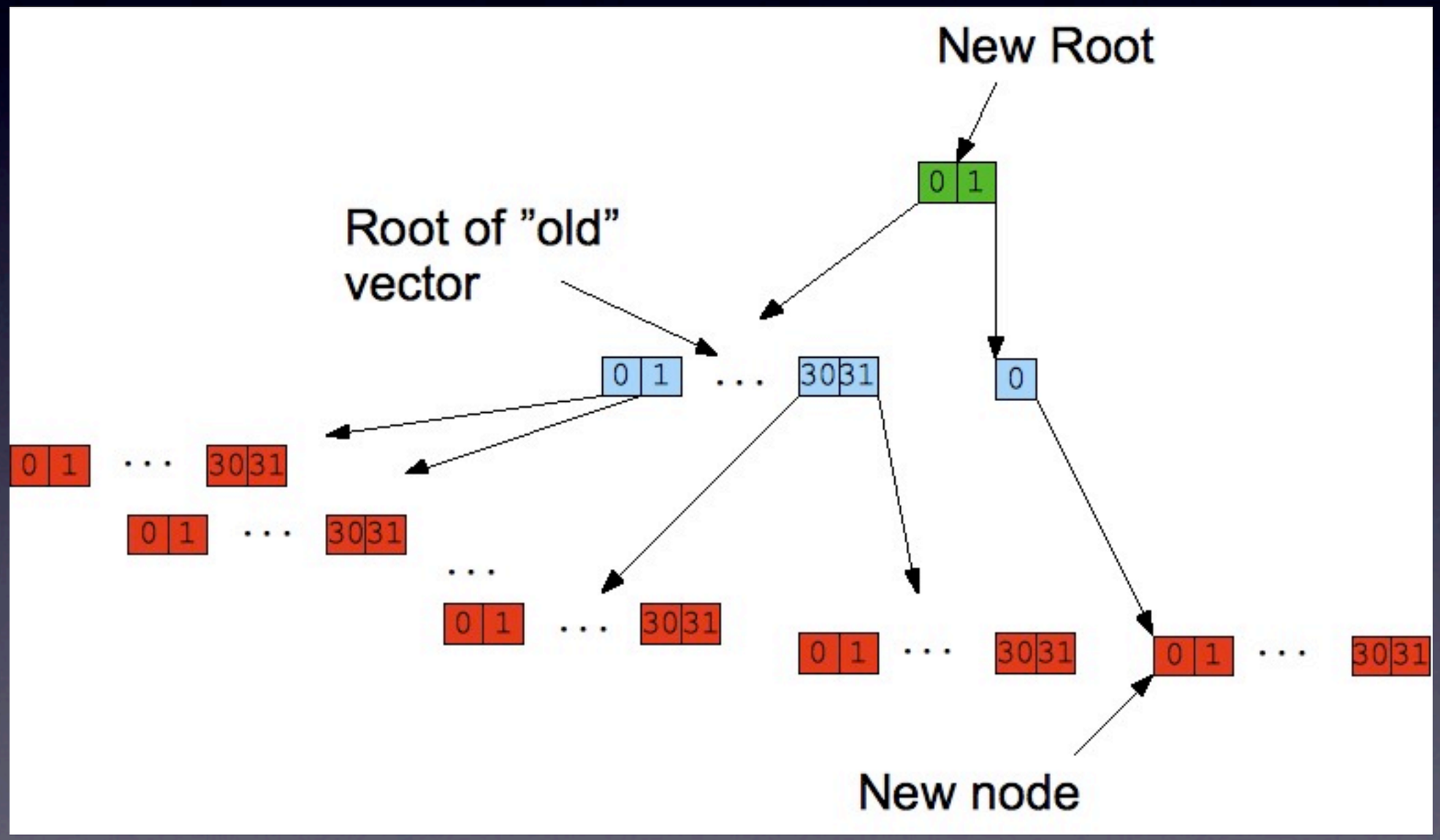
PersistentVector





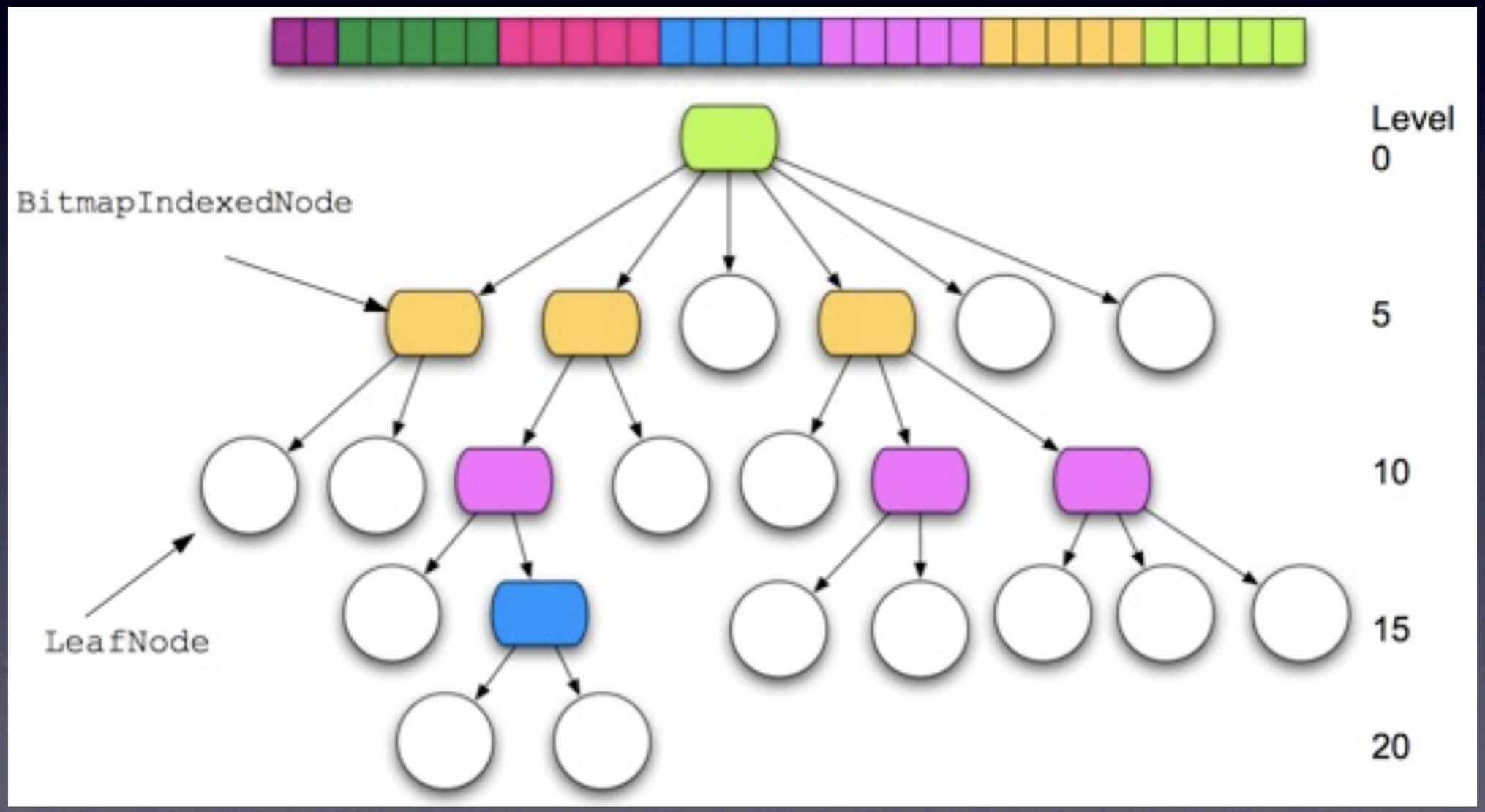
火龙果•整理
uml.org.cn

Insertion (put)

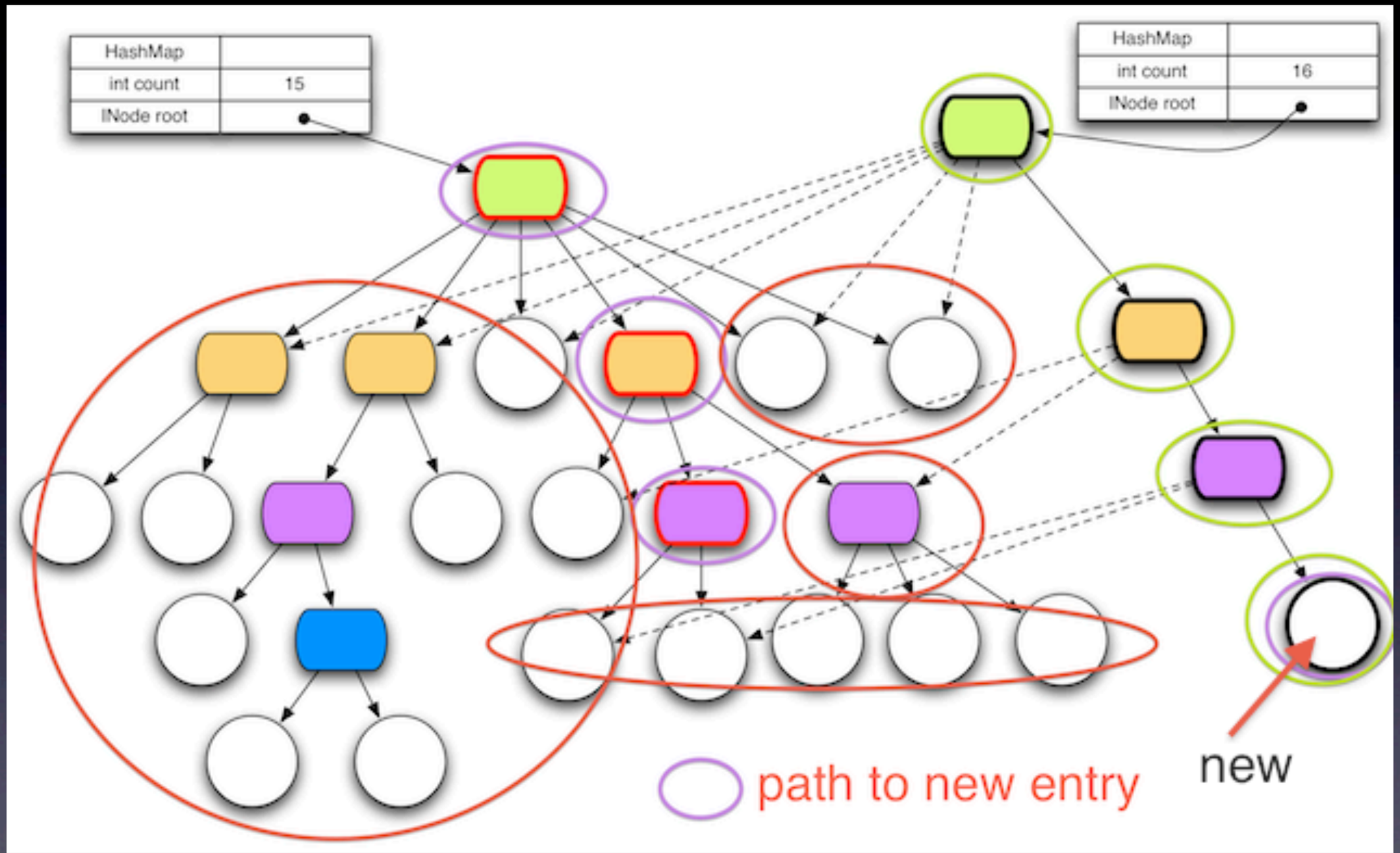


火龙果•整理
uml.org.cn

PersistentHashMap



火龙果•整理
uml.org.cn



火龙果•整理
uml.org.cn

并发编程模式

有状态(Stateful)	无状态(Stateless)
共享数据可变	共享数据不变
复杂，容易出错	简单
变量可能线程不安全	变量线程安全
不容易水平扩展	容易水平扩展

火龙果•整理
uml.org.cn

讨论

- 计算在座各位的年龄总和?

火龙果•整理
uml.org.cn

总结

- 划分任务
- 分析CPU和IO时间
- 设置合理的线程池大小
- 尽量用JUC包实现
- 避免共享的可变变量；尽量使用共享的不可改变变量或隔离的可变变量

火龙果•整理
uml.org.cn

网络服务编程框架

同步 Synchronous	异步 Asynchronous
阻塞	非阻塞
流程驱动	事件驱动
JDBC	libevent, Twisted, EventMachine, Node.JS, Java NIO