LEVEL 4 PUBLIC
Sony Computer Entertainment Europe

SCEE Research & Development

Sony Computer Entertainment Europe
Research & Development Division

Pitfalls of Object Oriented Programming

Tony Albrecht – Technical Consultant
Developer Services

PlayStation 3

What I will be covering

- A quick look at Object Oriented (OO) programming
- A common example
- Optimisation of that example
- Summary

Object Oriented (OO) Programming

- What is OO programming?
  - a programming paradigm that uses "objects" – data structures consisting of datafields and methods together with their interactions – to design applications and computer programs.
    (Wikipedia)

- Includes features such as
  - Data abstraction
  - Encapsulation
  - Polymorphism
  - Inheritance

What’s OOP for?

- OO programming allows you to think about problems in terms of objects and their interactions.
- Each object is (ideally) self contained
  – Contains its own code and data.
  – Defines an interface to its code and data.
- Each object can be perceived as a ‘black box’.

Objects

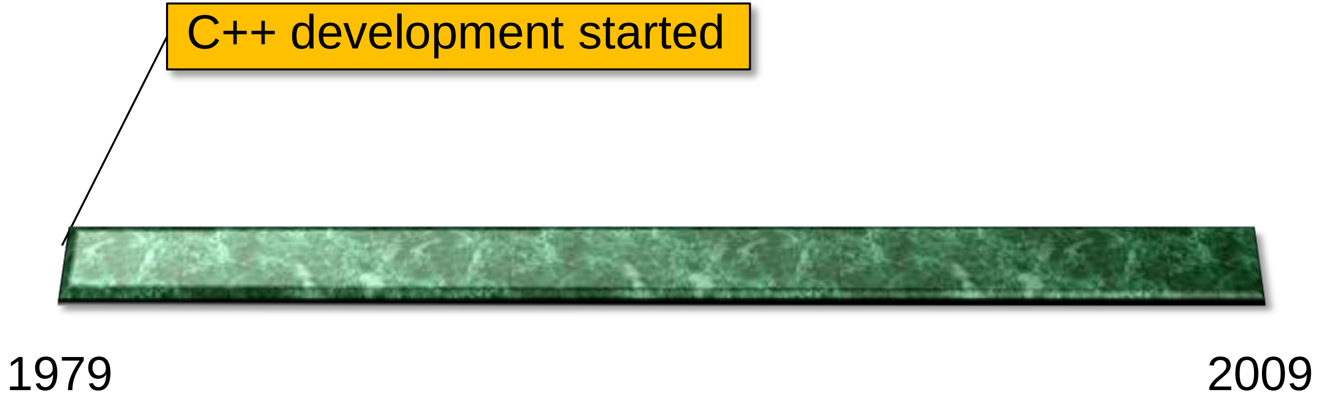
- If objects are self contained then they can be
  - Reused.
  - Maintained without side effects.
  - Used without understanding internal implementation/representation.

- This is good, yes?

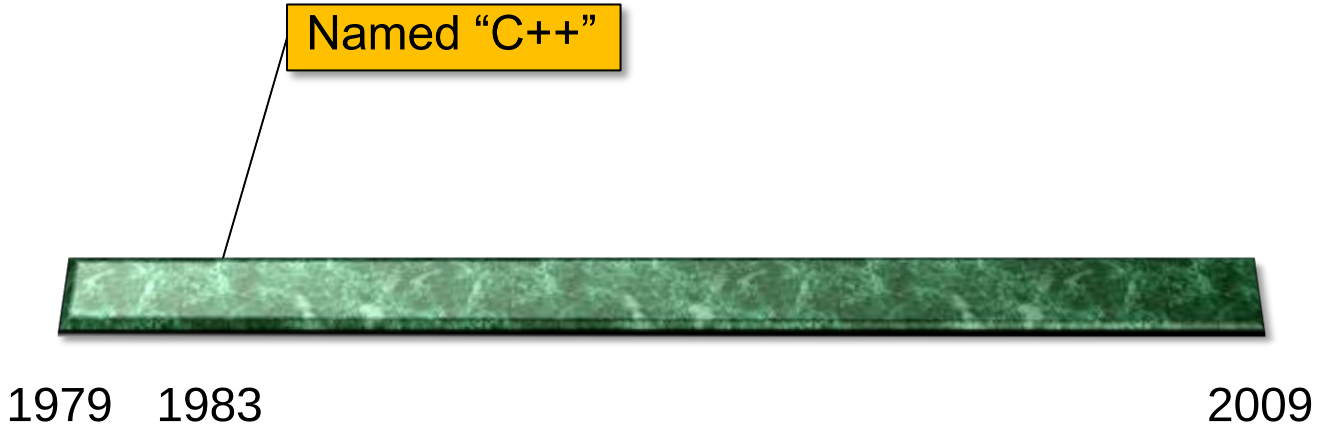
Are Objects Good?

- Well, yes
- And no.
- First some history.

A Brief History of C++

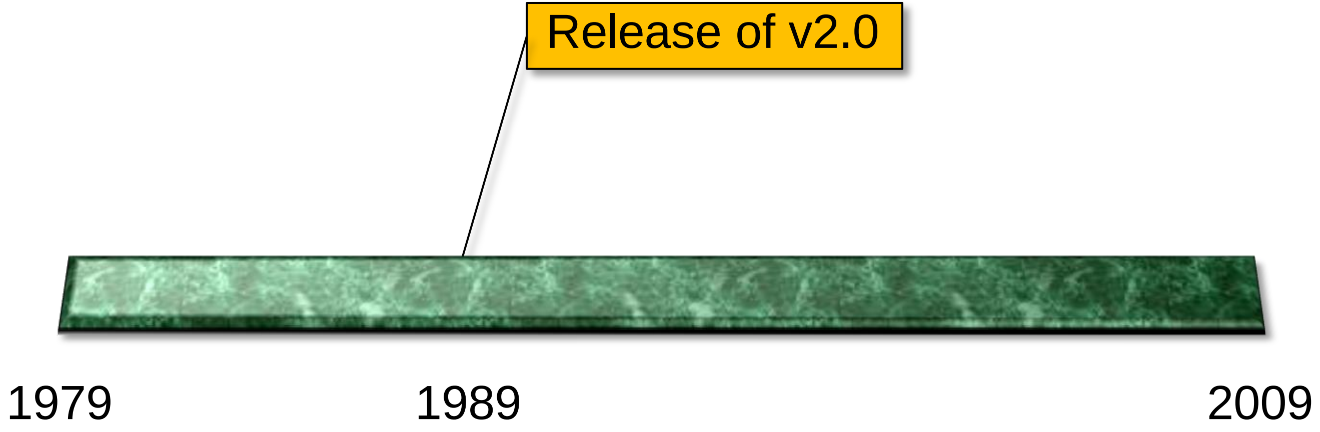


A Brief History of C++



A Brief History of C++

A Brief History of C++



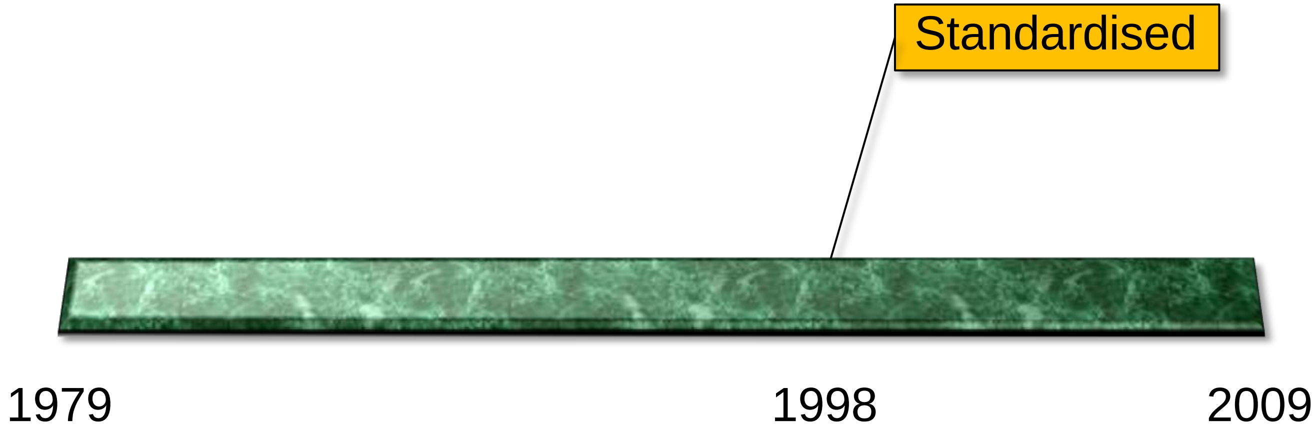
A Brief History of C++

Release c

Added
- multiple inheritance,
- abstract classes,
- static member functions,
- const member functions
- protected members.

1979 1989 2009

A Brief History of C++



A Brief History of C++

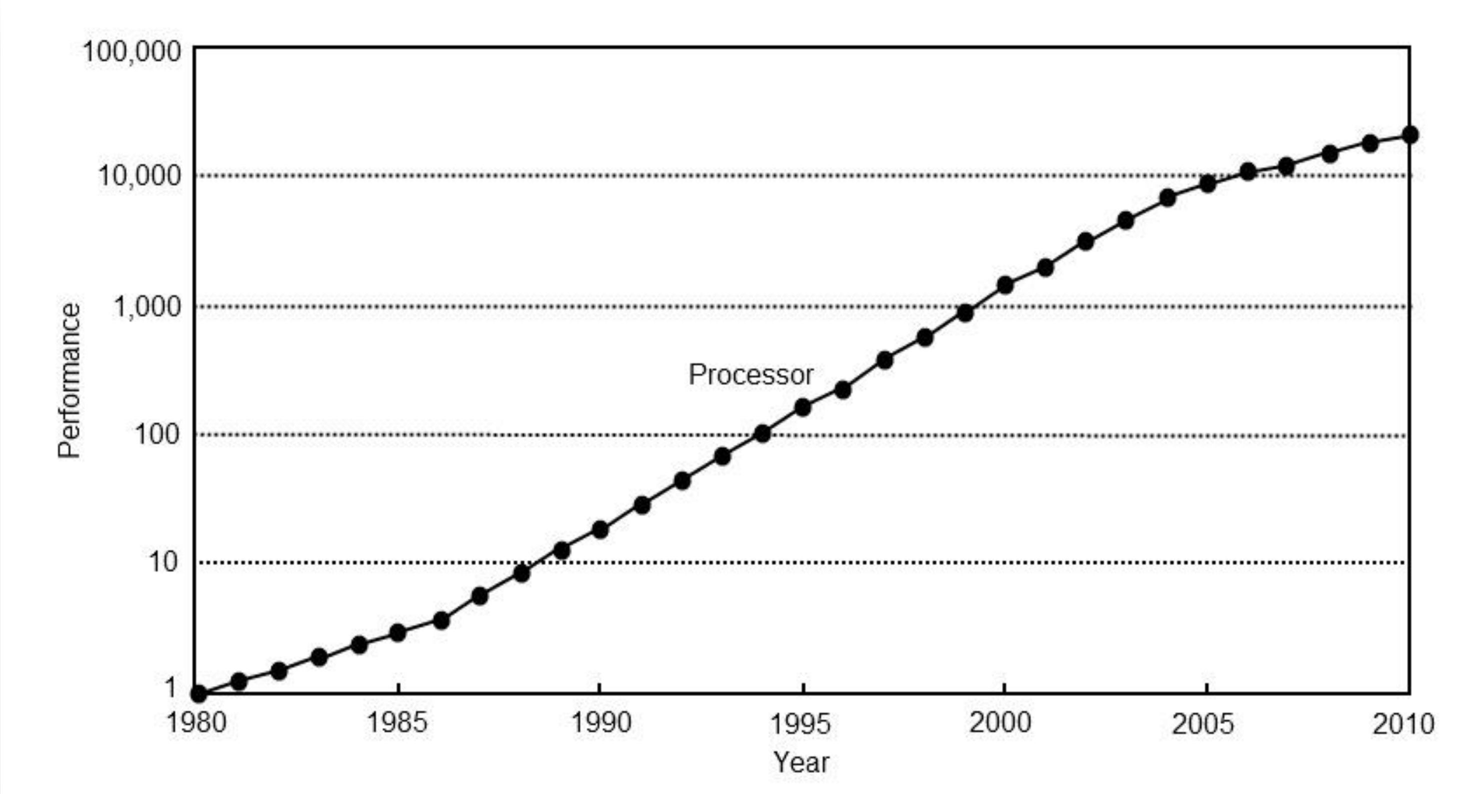
A Brief History of C++

So what has changed since 1979?

- Many more features have been added to C++
- CPUs have become much faster.
- Transition to multiple cores
- Memory has become faster.

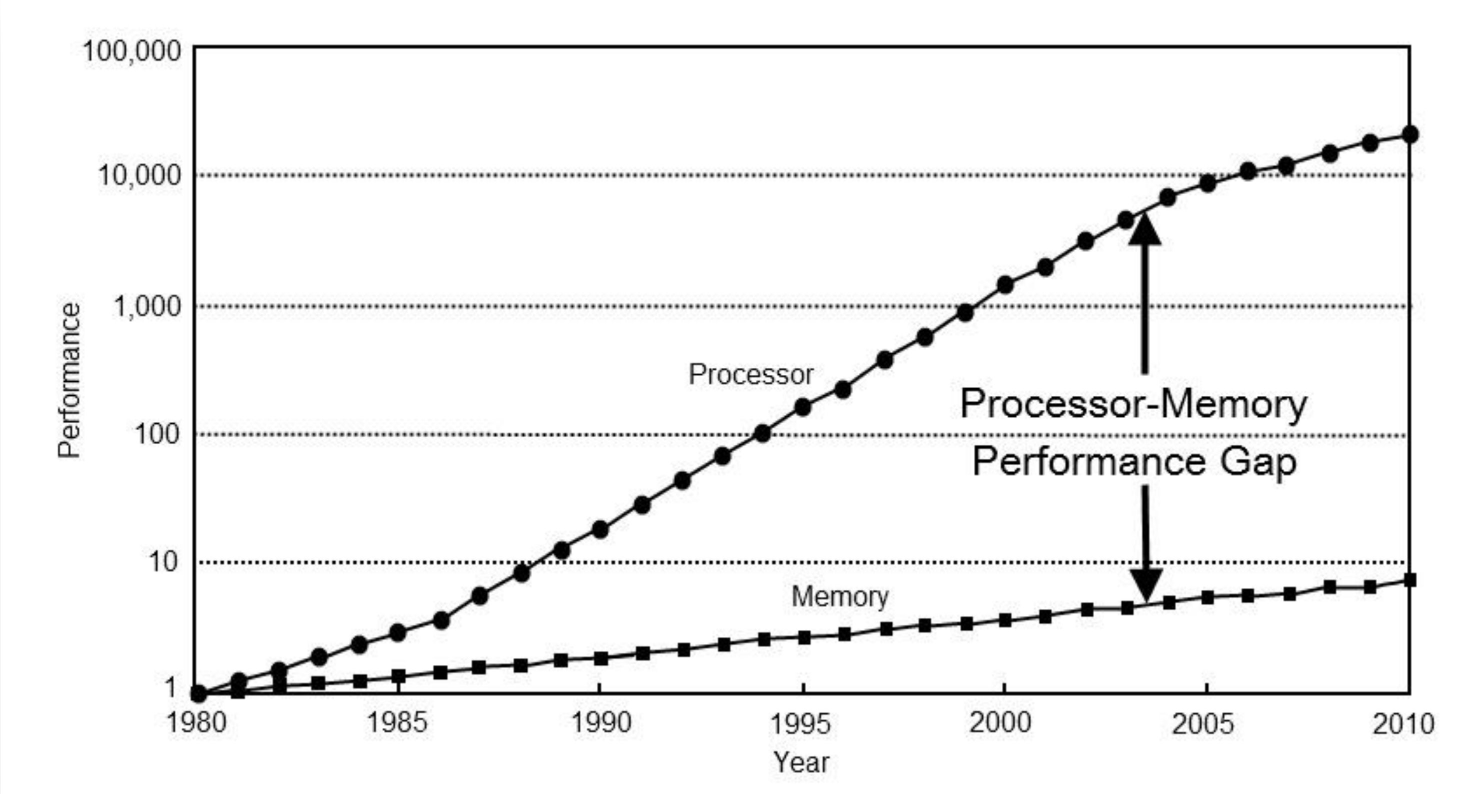
http://www.vintagecomputing.com

CPU performance



Computer architecture: a quantitative approach
By John L. Hennessy, David A. Patterson, Andrea C. Arpaci-Dusseau

CPU/Memory performance



Computer architecture: a quantitative approach
By John L. Hennessy, David A. Patterson, Andrea C. Arpaci-Dusseau

What has changed since 1979?

- One of the biggest changes is that memory access speeds are far slower (relatively)
  - 1980: RAM latency ~ 1 cycle
  - 2009: RAM latency ~ 400+ cycles

- What can *you* do in 400 cycles?

What has this to do with OO?

- OO classes encapsulate code and data.
- So, an instantiated object will generally contain all data associated with it.

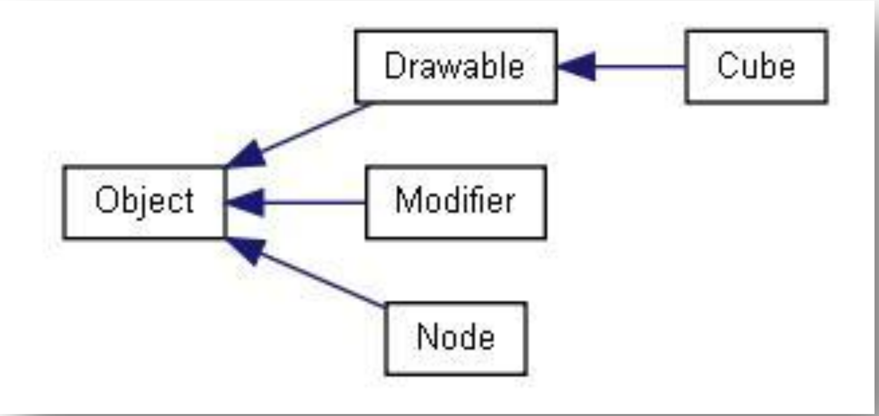
My Claim

- With modern HW (particularly consoles), excessive encapsulation is BAD.

- Data flow should be fundamental to your design (Data Oriented Design)

Consider a simple OO Scene Tree

- Base Object class
  - Contains general data
- Node
  - Container class
- Modifier
  - Updates transforms
- Drawable/Cube
  - Renders objects



Object

- Each object
  - Maintains bounding sphere for culling
  - Has transform (local and world)
  - Dirty flag (optimisation)
  - Pointer to Parent

```
class Object
{
    // <methods removed for clarity>
    Matrix4 m_Transform;
    Matrix4 m_WorldTransform;
    BoundingSphere m_BoundingSphere;
    BoundingSphere m_WorldBoundingSphere;
    char* m_Name;
    bool m_Dirty;
    Object* m_Parent;
};
```

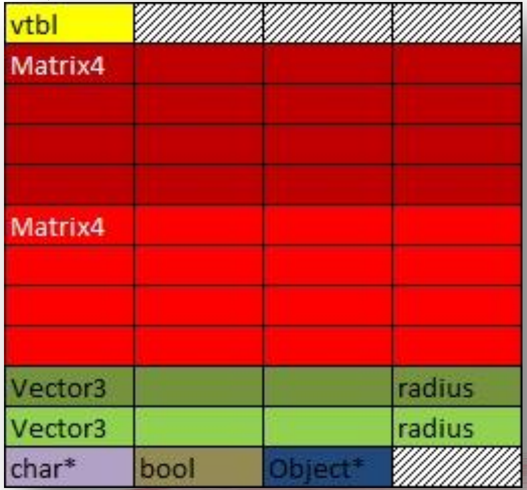
Objects

Class Definition

Each square is 4 bytes

Memory Layout

```
class Object
{
    // <methods removed for clarity>
    Matrix4 m_Transform;
    Matrix4 m_WorldTransform;
    BoundingSphere m_BoundingSphere;
    BoundingSphere m_WorldBoundingSphere;
    char* m_Name;
    bool m_Dirty;
    Object* m_Parent;
};
```



Nodes

- Each Node is an object, plus
  - Has a container of other objects
  - Has a visibility flag.

```
class Node : public Object
{
    // methods removed for clarity
    std::vector<Object*> m_Objects;
    bool m_IsVisible;
};
```

Nodes

Class Definition

```
class Node : public Object
{
    // methods removed for clarity
    std::vector<Object*> m_Objects;
    bool m_IsVisible;
};
```

Memory Layout

Consider the following code…

- Update the world transform and world space bounding sphere for each object.

```
const BoundingSphere& Node::GetWorldBoundingSphere(const Matrix4& parentTransform)
{
    m_WorldTransform = parentTransform*m_Transform;

    for(std::vector<Object*>::const_iterator itr = m_Objects.begin();
        itr!= m_Objects.end();
        ++itr)
    {
        m_WorldBoundingSphere.ExpandBy((*itr)->GetWorldBoundingSphere(m_WorldTransform));
    }
    return m_WorldBoundingSphere;
}
```

Consider the following code…

- Leaf nodes (objects) return transformed bounding spheres

```
virtual const BoundingSphere& GetWorldBoundingSphere(const Matrix4& parentTransform)
{
    if(m_Dirty)
        m_WorldBoundingSphere = m_BoundingSphere.Transform(parentTransform);
    return m_WorldBoundingSphere;
}
```

Consider the following code…

- Leaf nod … rn transformed bounding

What’s wrong with this code?

```
virtual const BoundingSphere& GetWorldBoundingSphere(const Matrix4& parentTransform)
{
    if(m_Dirty)
        m_WorldBoundingSphere = m_BoundingSphere.Transform(parentTransform);
    return m_WorldBoundingSphere;
}
```

Consider the following code…

- Leaf nod… formed bounding…

If m_Dirty=false then we get branch misprediction which costs 23 or 24 cycles.

```
virtual const BoundingSphere& GetWorldBoundingSphere(const Matrix4& parentTransform)
{
    if(m_Dirty)
        m_WorldBoundingSphere = m_BoundingSphere.Transform(parentTransform);
    return m_WorldBoundingSphere;
}
```

Consider the following code…

- Leaf nodes (objects) return transformed bounding

Calculation of the world bounding sphere takes only 12 cycles.

```
virtual const BoundingSphere& GetWorldBoundingSphere(c      Matrix4& parentTransform)
{
    if(m_Dirty)
        m_WorldBoundingSphere = m_BoundingSphere.Transform(parentTransform);
    return m_WorldBoundingSphere;
}
```

Consider the following code…

- Leaf nodes (objects) return transformed bounding

So using a dirty flag here is actually slower than not using one (in the case where it is false)

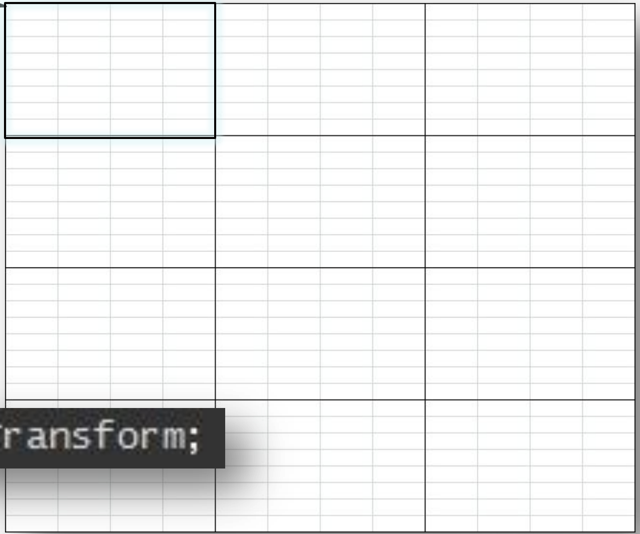
```
virtual const BoundingSphere& GetWorldBoundingSphere(const Matrix4& parentTransform)
{
    if(m_Dirty)
        m_WorldBoundingSphere = m_BoundingSphere.Transform(parentTransform);
    return m_WorldBoundingSphere;
}
```

Lets illustrate cache usage

Main Memory

Each cache line is 128 bytes

L2 Cache

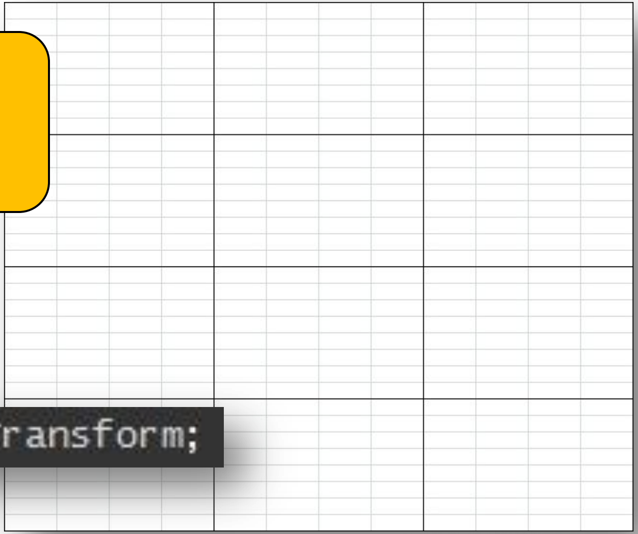


```
m_WorldTransform = parentTransform*m_Transform;
```

Cache usage

Main Memory

L2 Cache



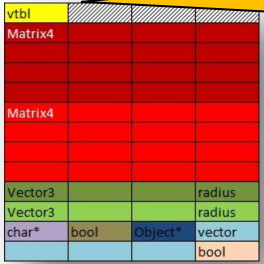
parentTransform is already in the cache (somewhere)

```
m_WorldTransform = parentTransform*m_Transform;
```

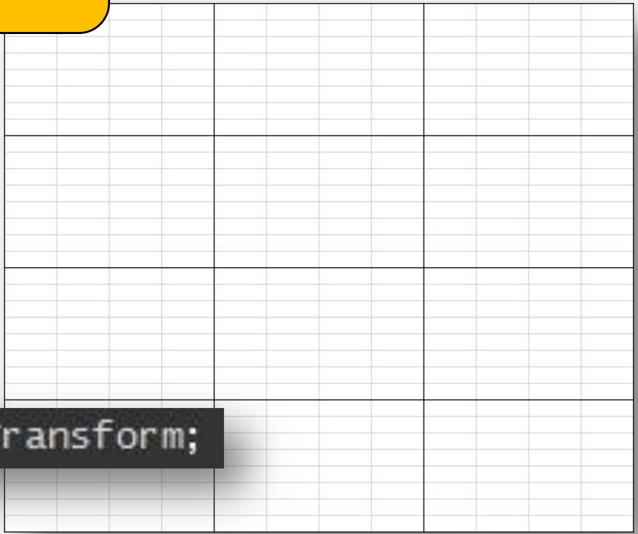
Cache usage

Assume this is a 128byte boundary (start of cacheline)

Main Memory



L2 Cache

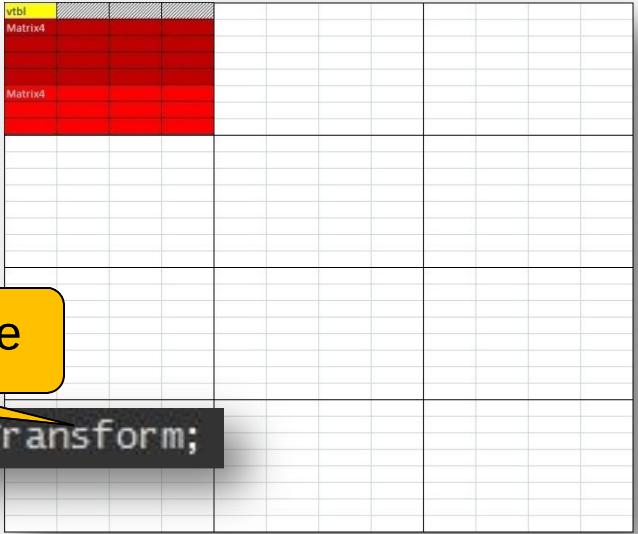


```
m_WorldTransform = parentTransform*m_Transform;
```

Cache usage

Main Memory

L2 Cache



Load m_Transform into cache

```
m_WorldTransform = parentTransform*m_Transform;
```

Cache usage

Main Memory

L2 Cache


```
m_WorldTransform = parentTransform*m_Transform;
```

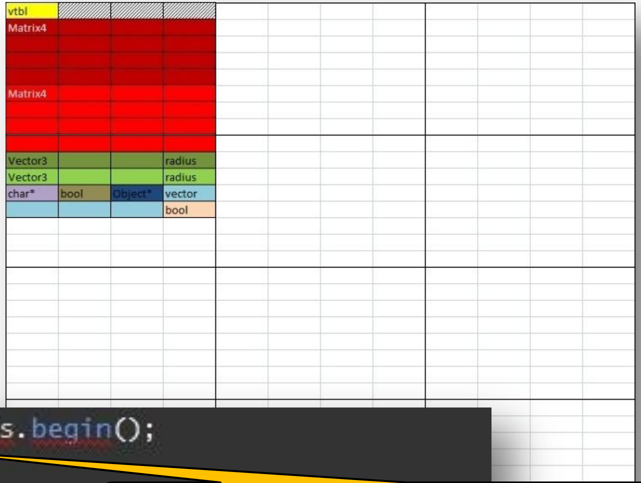
m_WorldTransform is stored via cache (write-back)

SCEE Research & Development SONY COMPUTER ENTERTAINMENT

Cache usage

Main Memory

L2 Cache



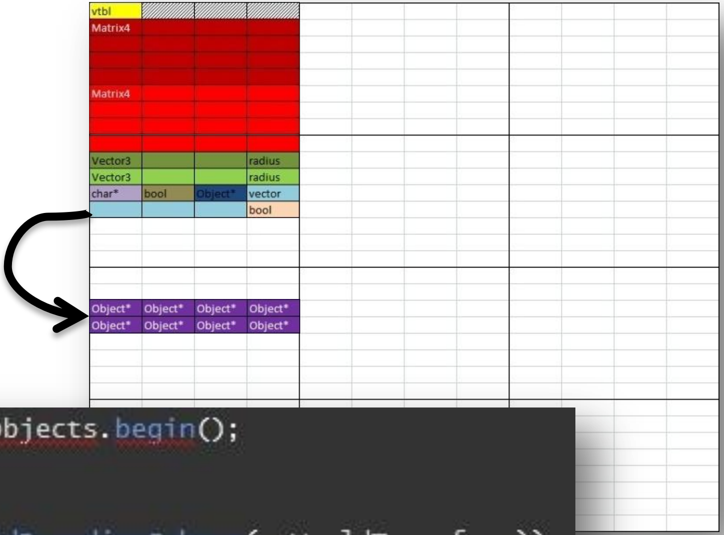

```
for(std::vector<Object*>::const_iterator itr = m_Objects.begin();
    itr!= m_Objects.end();
    ++itr)
{
    m_WorldBoundingSphere.ExpandBy((*itr)->GetWorldBoundingSp
}
```

Next it loads m_Objects

Cache usage

Main Memory

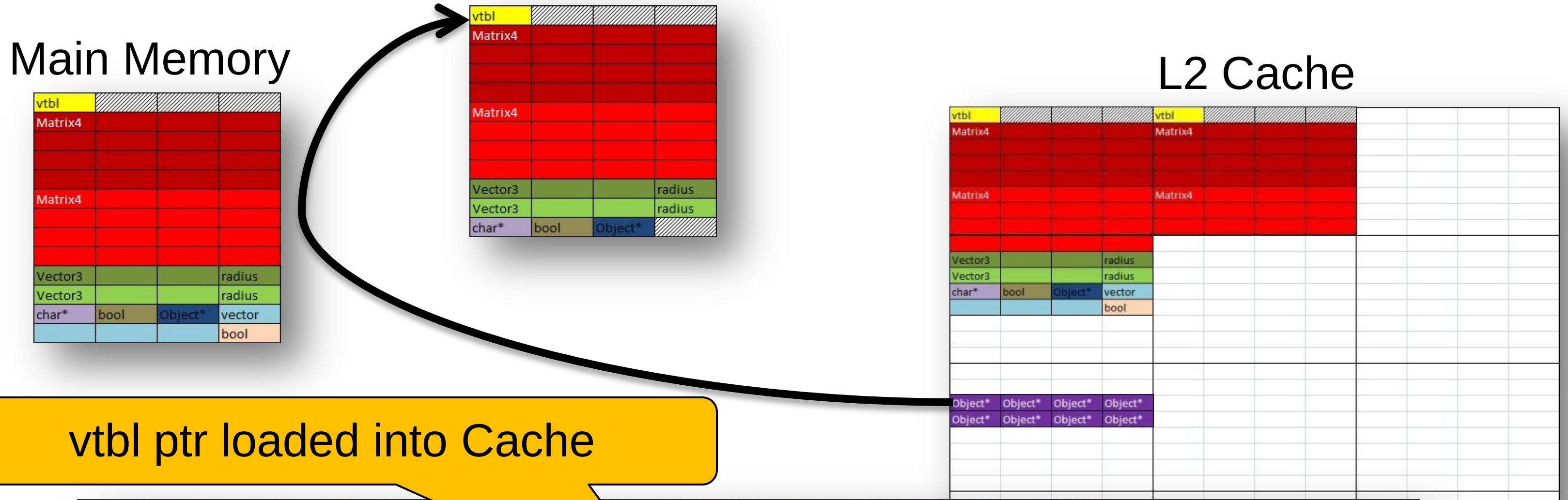
L2 Cache



Then a pointer is pulled from somewhere else (Memory managed by std::vector)

```
for(std::vector<Object*>::co        rator itr = m_Objects.begin();
    itr!= m_Objects.end();
    ++itr)
{
    m_WorldBoundingSphere.ExpandBy((*itr)->GetWorldBoundingSphere(m_WorldTransform));
}
```

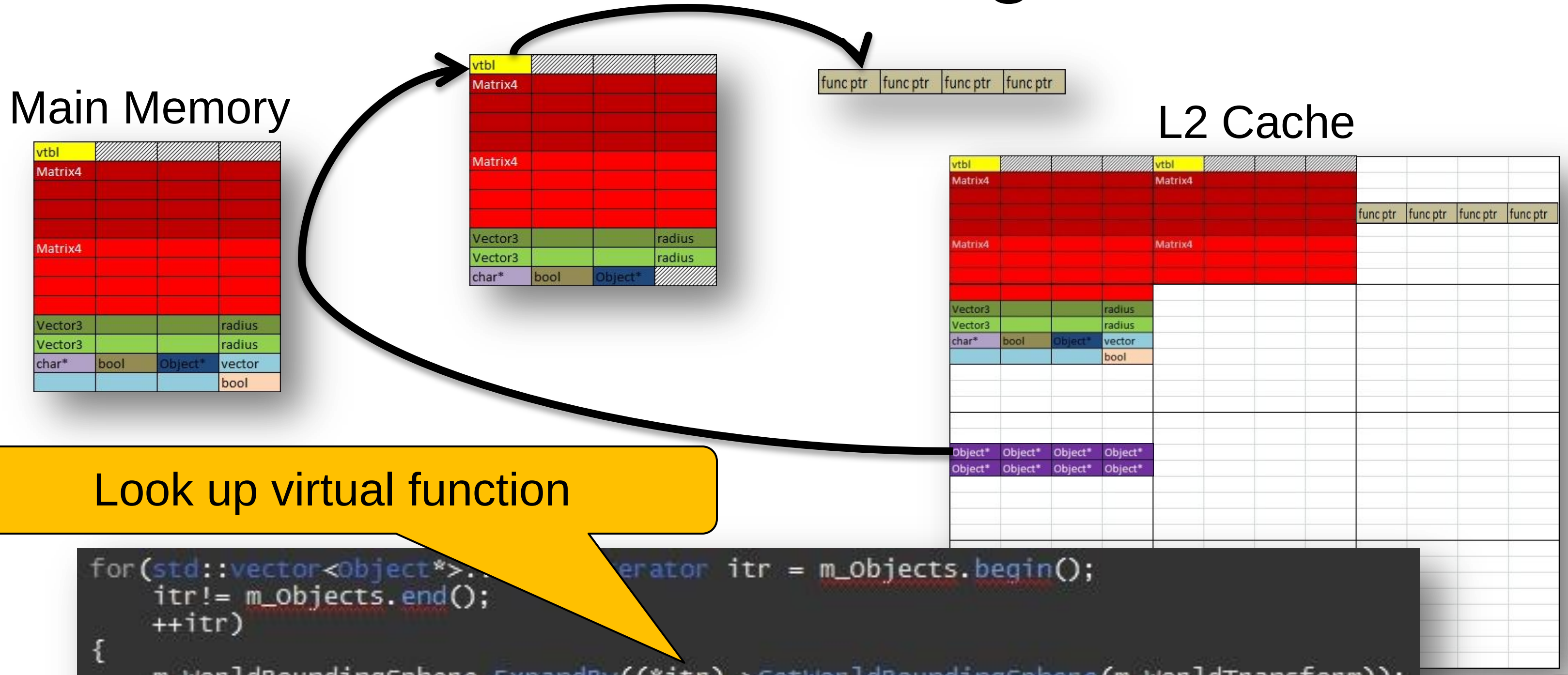
Cache usage



vtbl ptr loaded into Cache

```
for(std::vector<Object*>::iterator itr = m_Objects.begin();
    itr!= m_Objects.end();
    ++itr)
{
    m_WorldBoundingSphere.ExpandBy((*itr)->GetWorldBoundingSphere(m_WorldTransform));
}
```

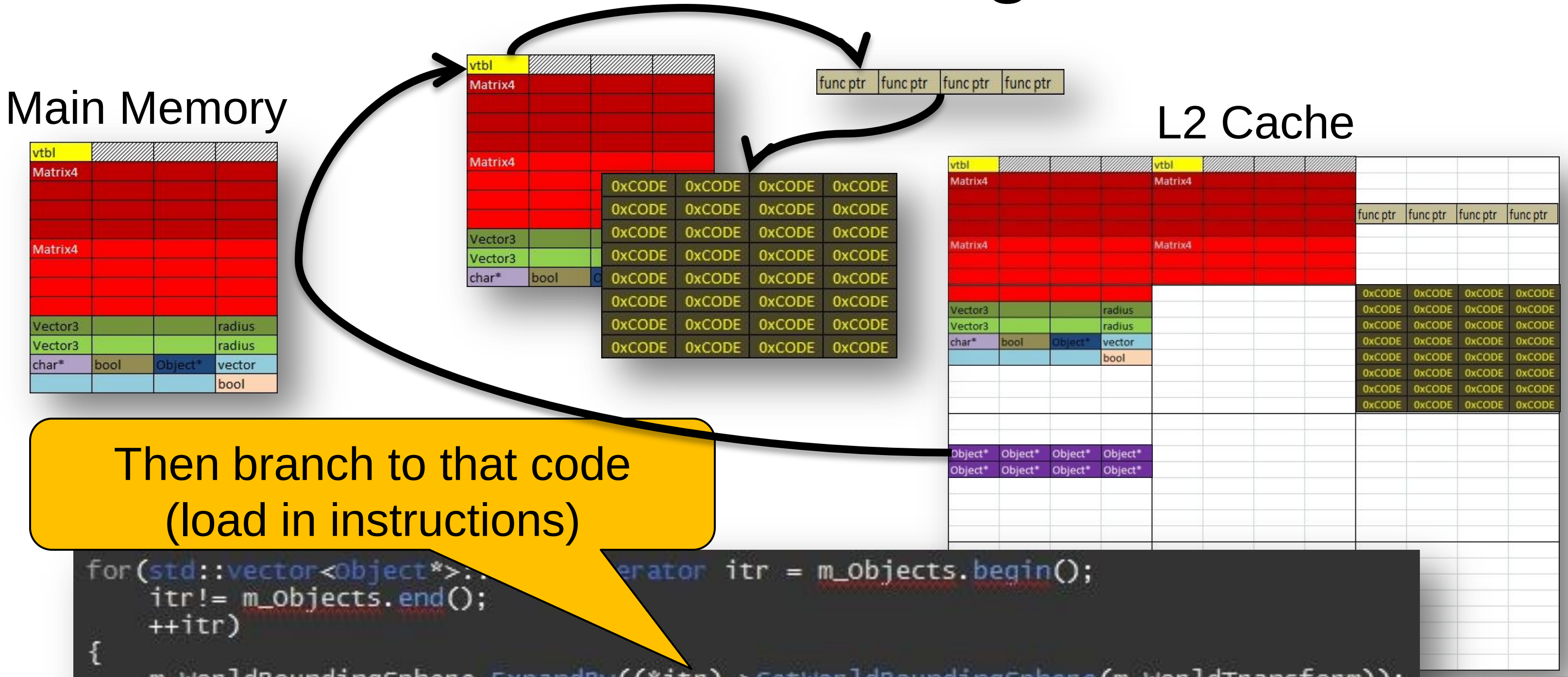
Cache usage



Look up virtual function

```
for(std::vector<Object*>::const_iterator itr = m_Objects.begin();
    itr!= m_Objects.end();
    ++itr)
{
    m_WorldBoundingSphere.ExpandBy((*itr)->GetWorldBoundingSphere(m_WorldTransform));
}
```

Cache usage



Then branch to that code
(load in instructions)

```
for(std::vector<Object*>::iterator itr = m_Objects.begin();
    itr!= m_Objects.end();
    ++itr)
{
    m_WorldBoundingSphere.ExpandBy((*itr)->GetWorldBoundingSphere(m_WorldTransform));
}
```

Cache usage

New code checks dirty flag then sets world bounding sphere

```
const BoundingSphere& GetWorldBoundingSphere(const Matrix4& parentTransform)
{
    if(m_Dirty)
        m_WorldBoundingSphere = m_BoundingSphere.Transform(parentTransform);
    return m_WorldBoundingSphere;
}
```

Cache usage

Node’s World Bounding Sphere is then Expanded

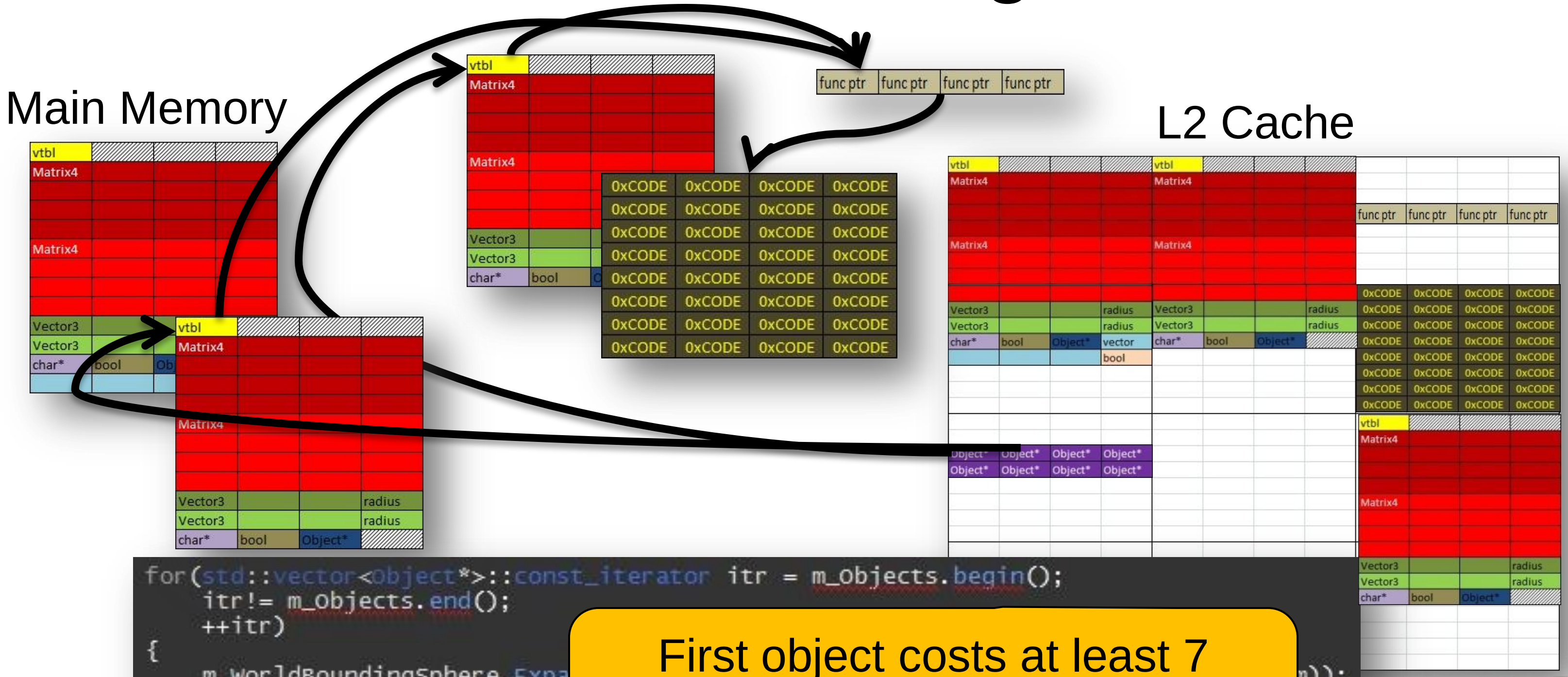
```
for(std::vector<Object*>::const_iterator itr = m_Objects.begin();
    itr!= m_Objects.end();
    ++itr)
{
    m_WorldBoundingSphere.ExpandBy((*itr)->GetWorldBoundingSphere(m_WorldTransform));
}
```

Cache usage


```
for(std::vector<Object*>::const_iterator itr = m_Objects.begin();
    itr!= m_Objects.end();
    ++itr)
{
    m_WorldBoundingSphere.Expa                                    m));
}
```

Then the next Object is processed

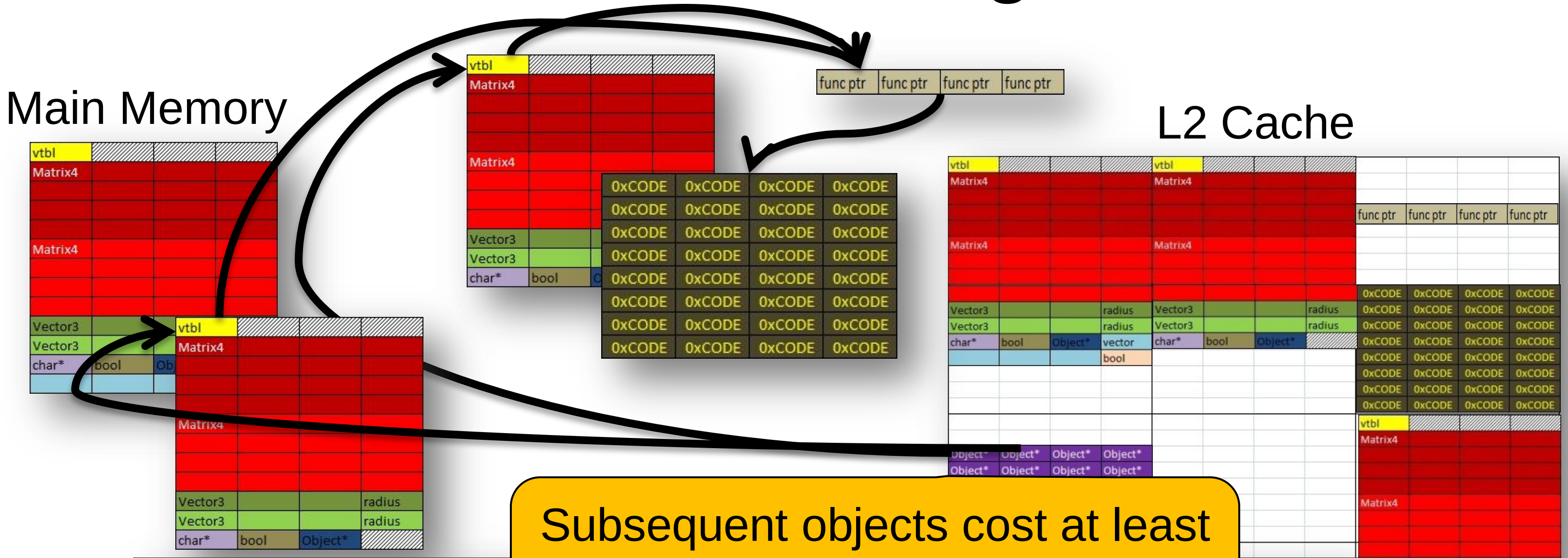
Cache usage




```
for(std::vector<Object*>::const_iterator itr = m_Objects.begin();
    itr!= m_Objects.end();
    ++itr)
{
    m_WorldBoundingSphere.Expa                                m));
}
```

First object costs at least 7 cache misses

Cache usage

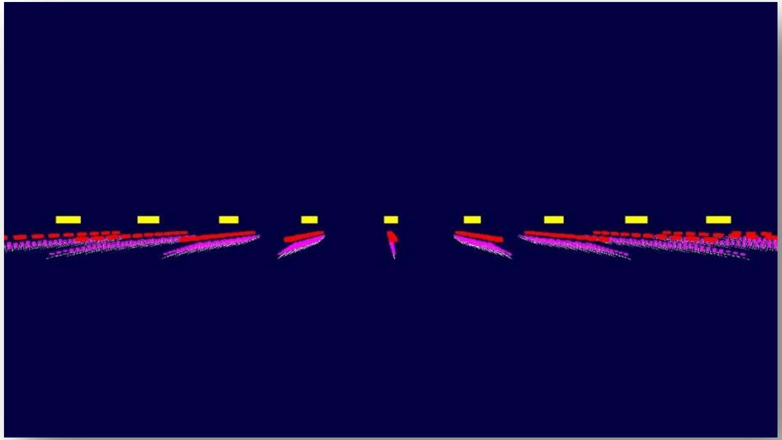



```
for(std::vector<Object*>::
    itr!= m_Objects.end();
    ++itr)
{
    m_WorldBoundingSphere.ExpandBy((*itr)->GetWorldBoundingSphere(m_WorldTransform));
}
```

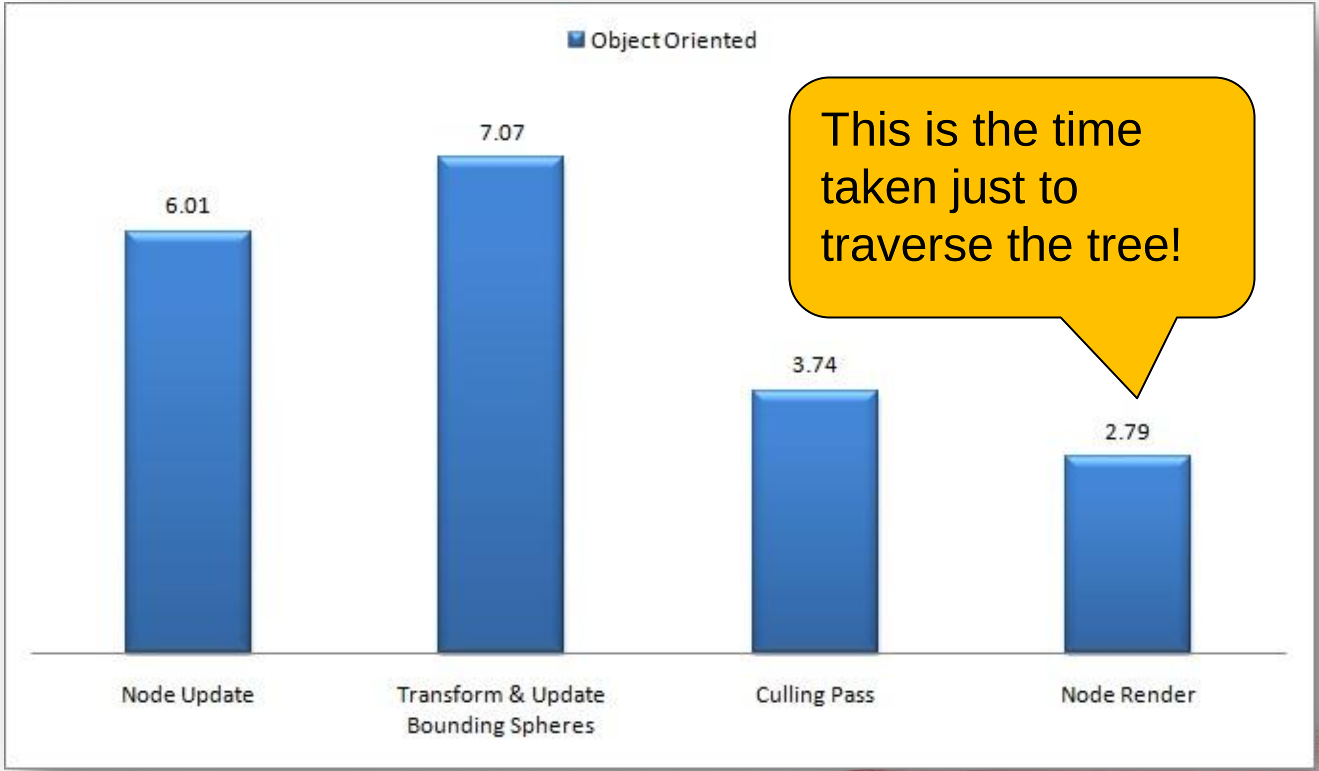
Subsequent objects cost at least 2 cache misses each

The Test

- 11,111 nodes/objects in a tree 5 levels deep
- Every node being transformed
- Hierarchical culling of tree
- Render method is empty



Performance

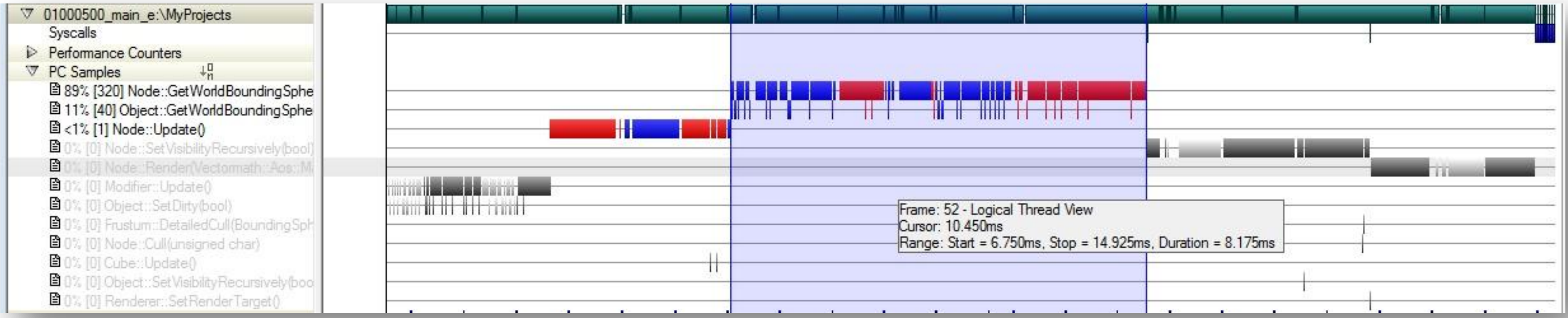


This is the time taken just to traverse the tree!

Why is it so slow?

~22ms

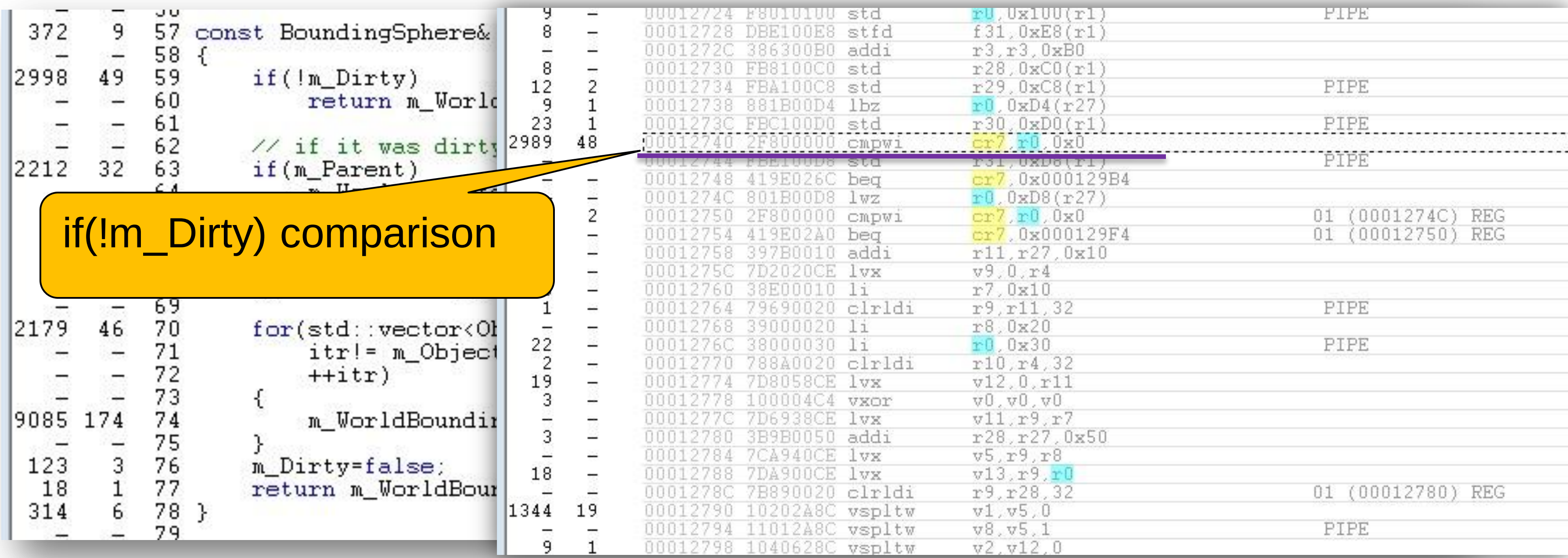
Look at GetWorldBoundingSphere()



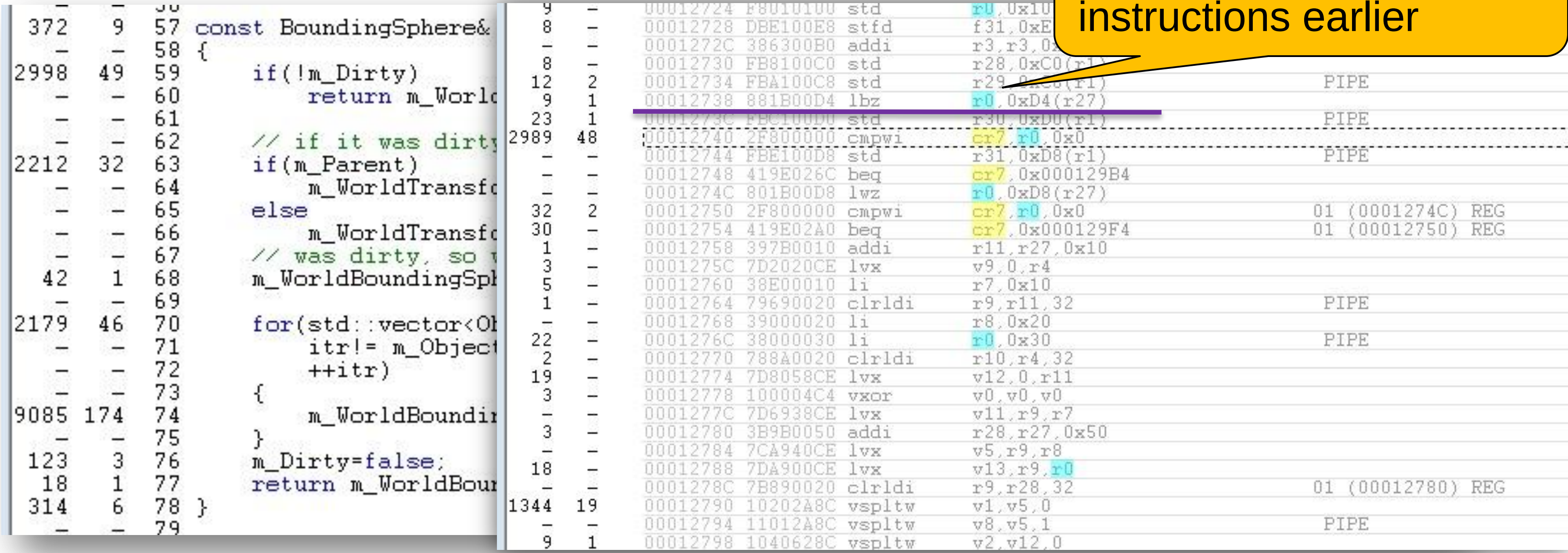
Samples can be a little misleading at the source code level

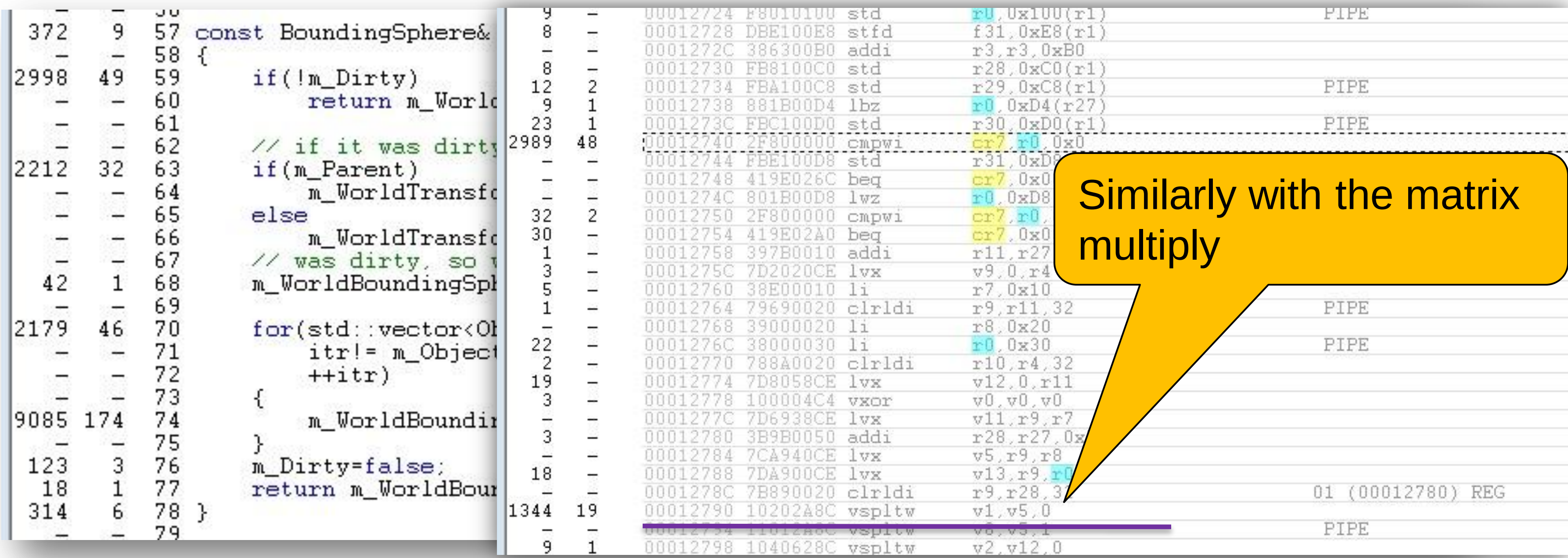
```
 372    9  const BoundingSphere& Node::GetWorldBoundingSphere(const Matrix4& parentTransform)
    –    –  {
2998   49      if(!m_Dirty)
    –    –          return m_WorldBoundingSphere;
    –    –
    –    –      // if it was dirty, then we need to update the bounding volumes and transforms
2212   32      if(m_Parent)
    –    –          m_WorldTransform = parentTransform*m_Transform;
    –    –      else
    –    –          m_WorldTransform=m_Transform;
    –    –      // was dirty, so we need to recalculate the bounds of the children
  42    1      m_WorldBoundingSphere=BoundingSphere(); // zero it
    –    –
2179   46      for(std::vector<Object*>::const_iterator itr = m_Objects.begin();
    –    –          itr!= m_Objects.end();
    –    –          ++itr)
    –    –      {
9085  174          m_WorldBoundingSphere.ExpandBy((*itr)->GetWorldBoundingSphere(m_WorldTransform));
    –    –      }
 123    3      m_Dirty=false;
  18    1      return m_WorldBoundingSphere;
 314    6  }
```

SCEE Research & Development



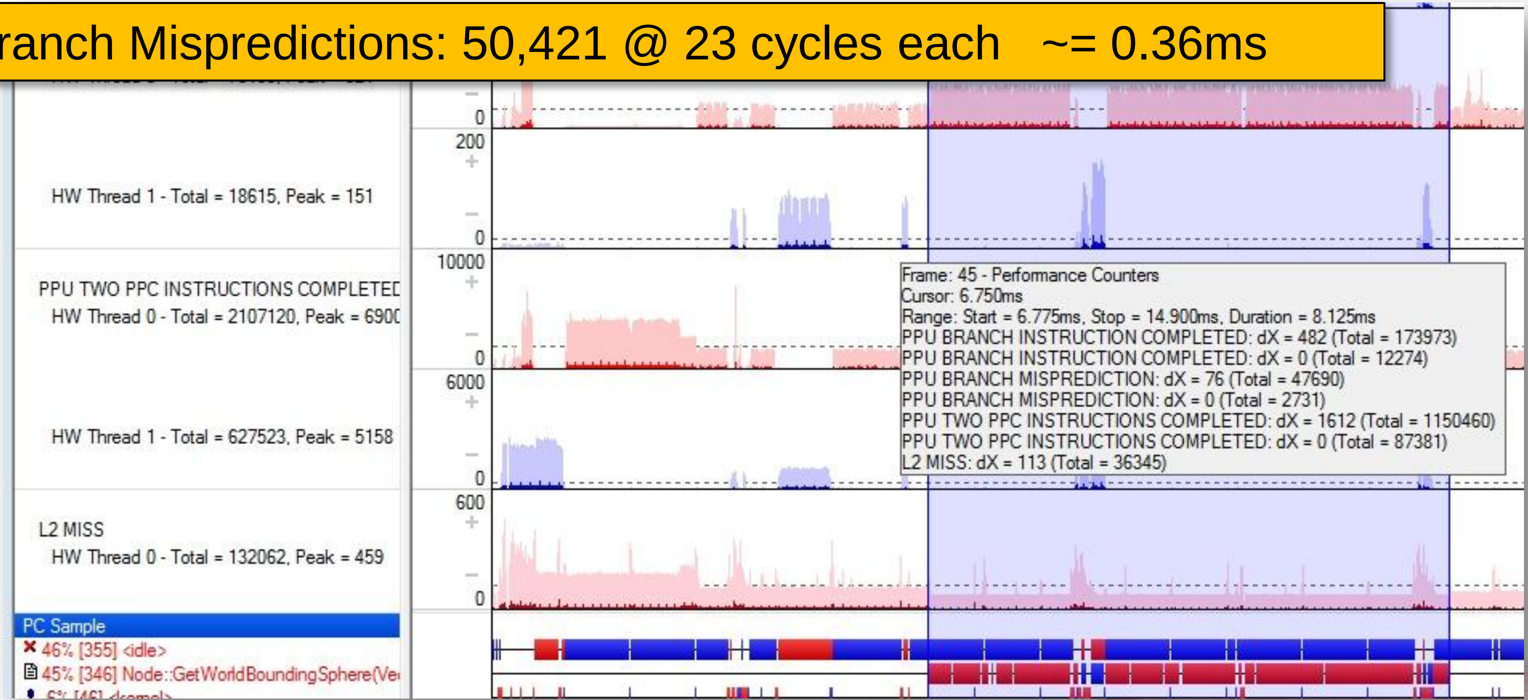
Stalls due to the load 2 instructions earlier




372 9 57 const BoundingSphere&
– – 58 {
2998 49 59 if(!m_Dirty)
– – 60 return m_Worl
– – 61
– – 62 // if it was dirty
2212 32 63 if(m_Parent)
– – 64 m_WorldTransf
– – 65 else
– – 66 m_WorldTransf
– – 67 // was dirty, so
42 1 68 m_WorldBoundingSph
– – 69
2179 46 70 for(std::vector<Ob
– – 71 itr!= m_Object
– – 72 ++itr)
– – 73 {
9085 174 74 m_WorldBoundin
– – 75 }
123 3 76 m_Dirty=false;
18 1 77 return m_WorldBour
314 6 78 }
8 – 00012728 DBE100E8 stfd f31,0xE8(r1)
– – 0001272C 386300B0 addi r3,r3,0xB0
8 – 00012730 FB8100C0 std r28,0xC0(r1)
12 2 00012734 FBA100C8 std r29,0xC8(r1) PIPE
9 1 00012738 881B00D4 lbz r0,0xD4(r27)
23 1 0001273C FBC100D0 std r30,0xD0(r1) PIPE
2989 48 00012740 2F800000 cmpwi cr7,r0,0x0
– – 00012744 FBE100D8 std r31,0xD
– – 00012748 419E026C beq cr7,0x0
– – 0001274C 801B00D8 lwz r0,0xD8
32 2 00012750 2F800000 cmpwi cr7,r0,
30 – 00012754 419E02A0 beq cr7,0x0
1 – 00012758 397B0010 addi r11,r27
3 – 0001275C 7D2020CE lvx v9,0,r4
5 – 00012760 38E00010 li r7,0x10
1 – 00012764 79690020 clrldi r9,r11,32 PIPE
– – 00012768 39000020 li r8,0x20
22 – 0001276C 38000030 li r0,0x30 PIPE
2 – 00012770 788A0020 clrldi r10,r4,32
19 – 00012774 7D8058CE lvx v12,0,r11
3 – 00012778 100004C4 vxor v0,v0,v0
– – 0001277C 7D6938CE lvx v11,r9,r7
3 – 00012780 3B9B0050 addi r28,r27,0x
– – 00012784 7CA940CE lvx v5,r9,r8
18 – 00012788 7DA900CE lvx v13,r9,r0
– – 0001278C 7B890020 clrldi r9,r28,3 01 (00012780) REG
1344 19 00012790 10202A8C vspltw v1,v5,0
PIPE
9 1 00012798 1040628C vspltw v2,v12,0
Similarly with the matrix multiply

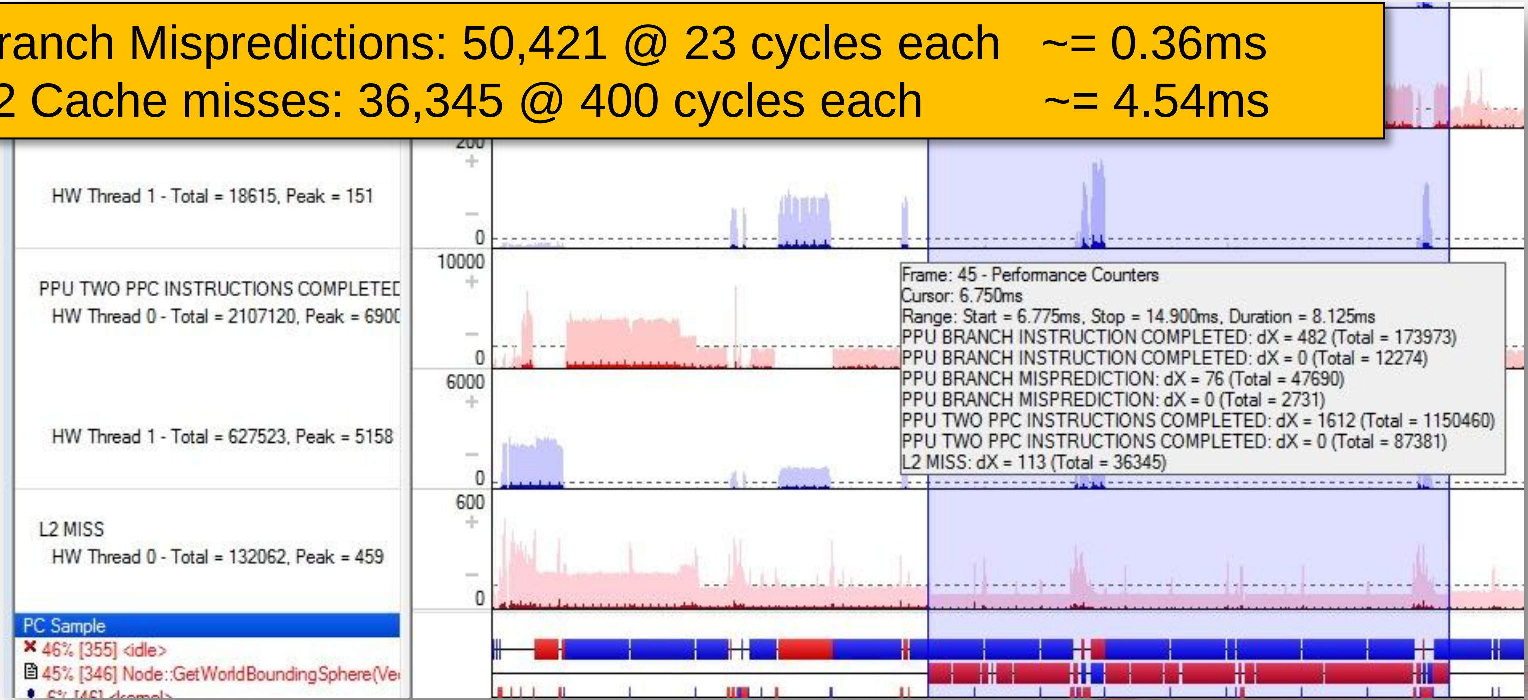
Some rough calculations

Branch Mispredictions: 50,421 @ 23 cycles each ~= 0.36ms



Some rough calculations

Branch Mispredictions: 50,421 @ 23 cycles each ~= 0.36ms
L2 Cache misses: 36,345 @ 400 cycles each ~= 4.54ms



- From Tuner, ~ 3 L2 cache misses per object
  - These cache misses are mostly sequential (more than 1 fetch from main memory can happen at once)
  - Code/cache miss/code/cache miss/code…

Slow memory is the problem here

- How can we fix it?
- And still keep the same functionality and interface?

The first step

- Use homogenous, sequential sets of data

```
class Object
{
    // <methods removed for clarity>
    Matrix4 *m_Transform;
    Matrix4 *m_WorldTransform;
    BoundingSphere *m_BoundingSphere;
    BoundingSphere *m_WorldBoundingSphere;
    char* m_Name;
    bool m_Dirty;
    Object* m_Parent;
};
```

vtbl	Matrix4*	Matrix4*	Bsphere*
Bsphere*	char*	bool	Object*
vector			

```
class Node : public Object
{
    // methods removed for clarity
    std::vector<Object*> m_Objects;
    bool m_IsVisible;
};
```

Homogeneous Sequential Data

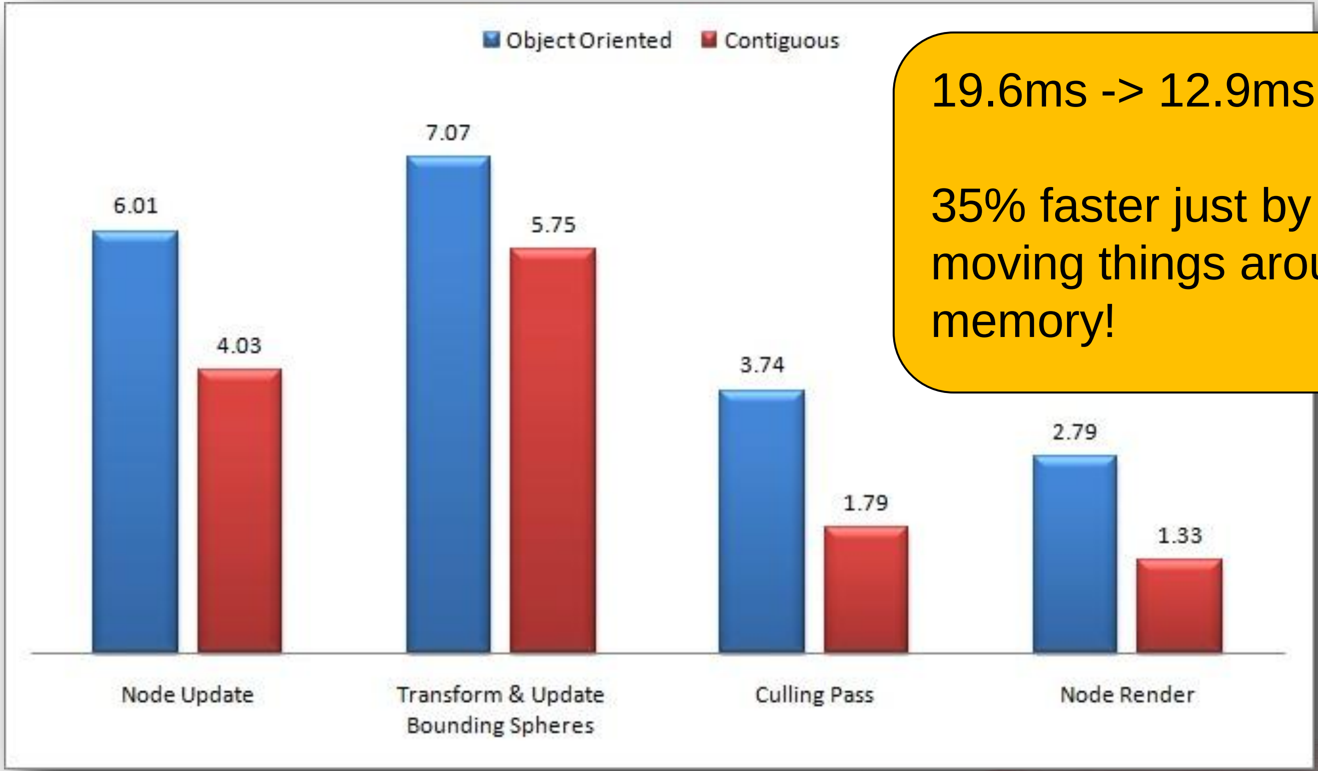
vtbl	Matrix4*	Matrix4*	Bsphere*
Bsphere*	char*	bool	Object*
vector			

World transforms	Local Transforms	World Bounding Spheres	Local Bounding Spheres
Matrix4	Matrix4	Vector3	Vector3
		Vector3	Vector3
		Vector3	Vector3
		Vector3	Vector3
Matrix4	Matrix4	Vector3	Vector3
		Vector3	Vector3
		Vector3	Vector3
		Vector3	Vector3
Matrix4	Matrix4		
Matrix4	Matrix4		

Generating Contiguous Data

- Use custom allocators
  - Minimal impact on existing code
- Allocate contiguous
  - Nodes
  - Matrices
  - Bounding spheres

Performance



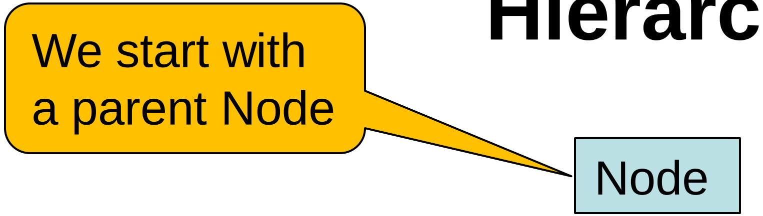
19.6ms -> 12.9ms

35% faster just by moving things around in memory!

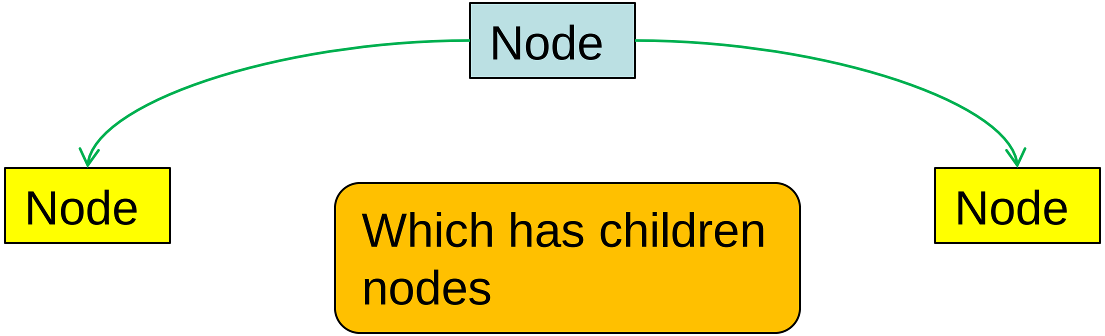
What next?

- Process data in order
- Use implicit structure for hierarchy
  - Minimise to and fro from nodes.
- Group logic to optimally use what is already in cache.
- Remove regularly called virtuals.

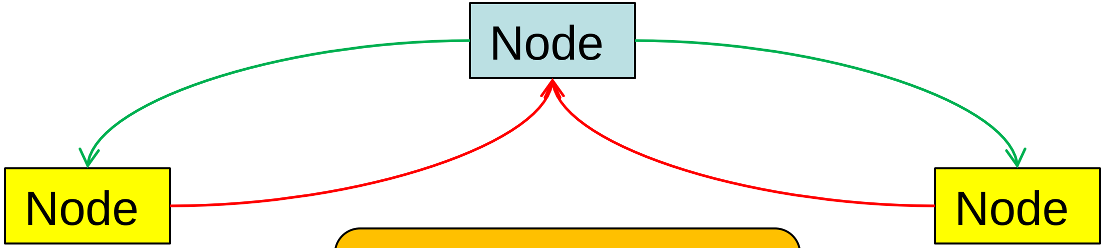
Hierarchy



Hierarchy



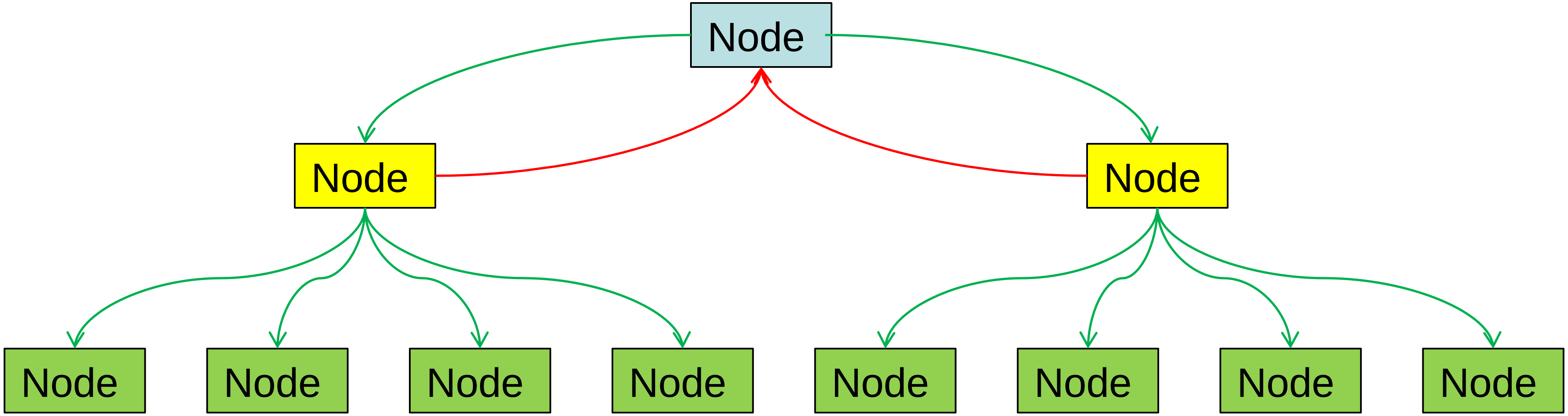
Hierarchy



And they have a parent

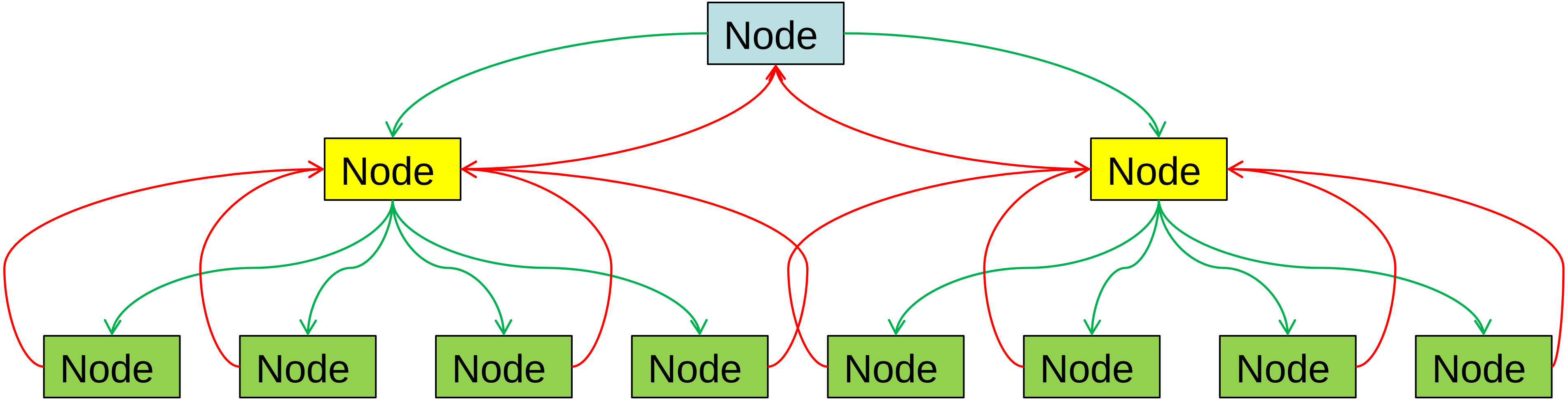
SCEE Research & Development

Hierarchy



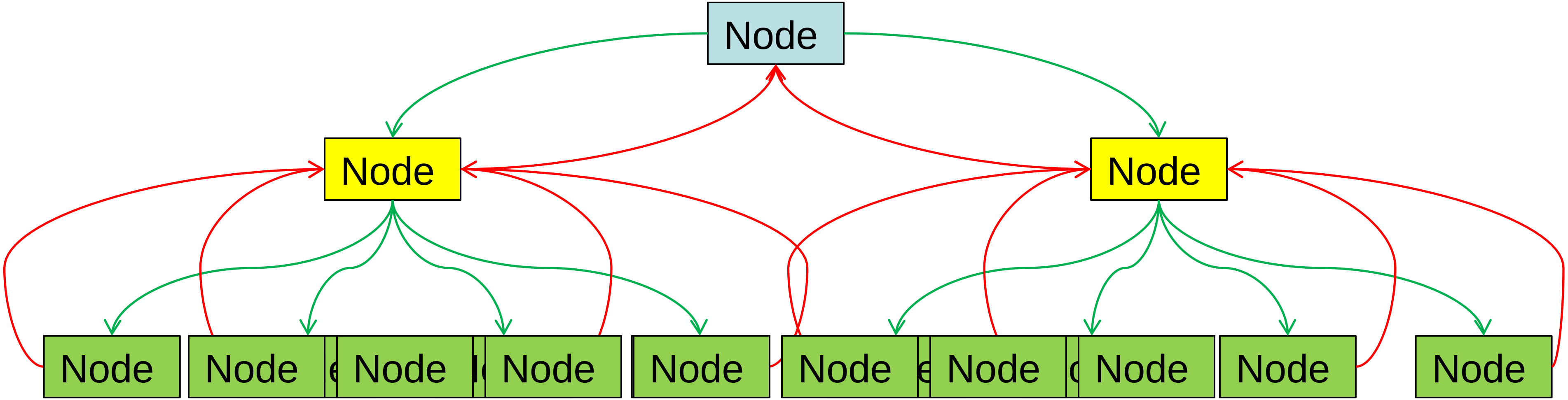
And they have children

Hierarchy



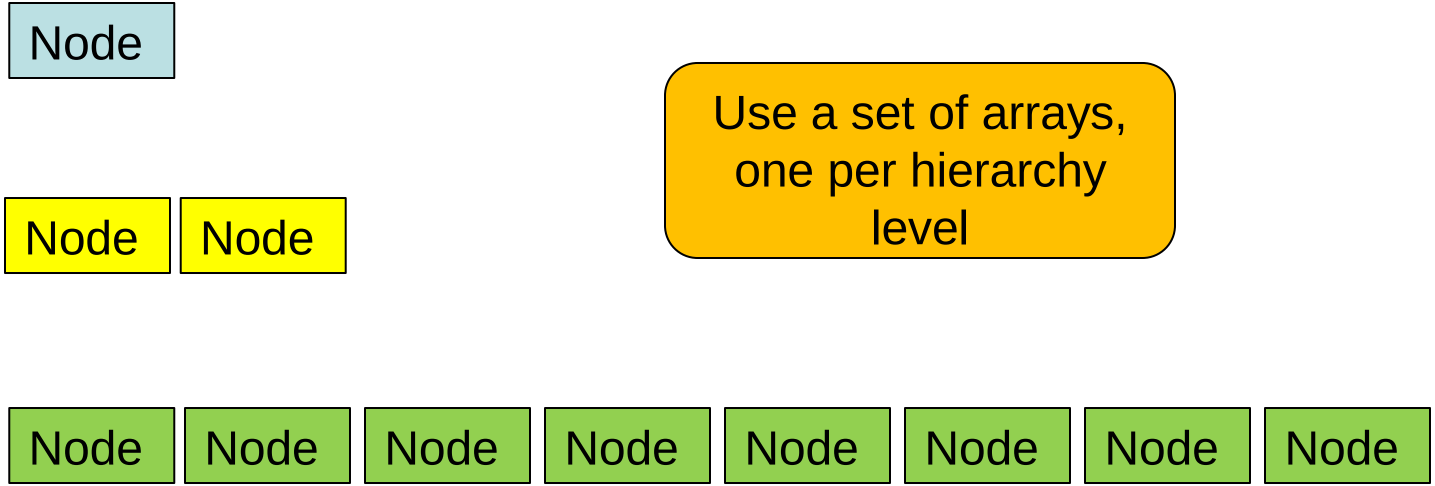
And they all have parents

Hierarchy



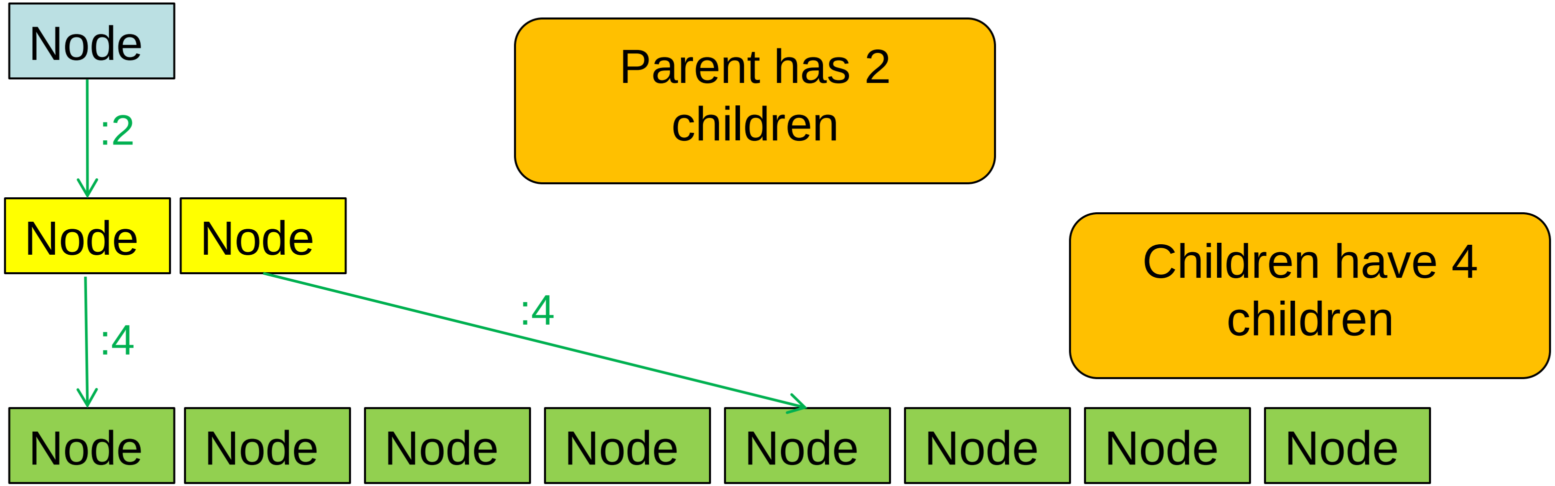
A lot of this information can be inferred

Hierarchy

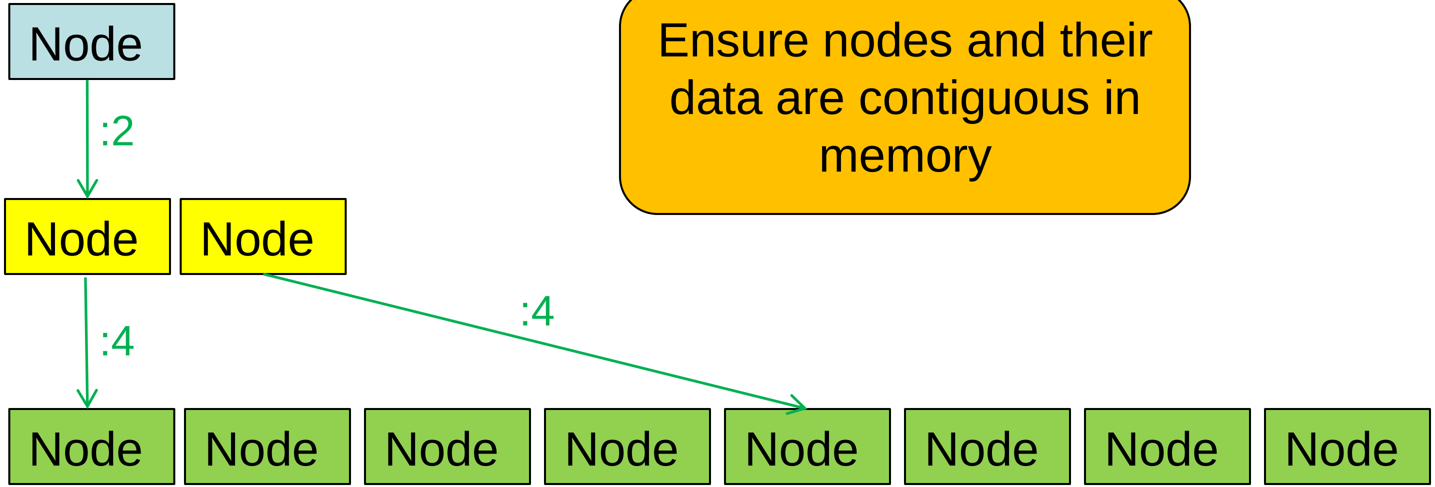


Use a set of arrays, one per hierarchy level

Hierarchy


Node
:2
Node
Node
:4
:4
Node
Node
Node
Node
Node
Node
Node
Node
Parent has 2 children
Children have 4 children

Hierarchy

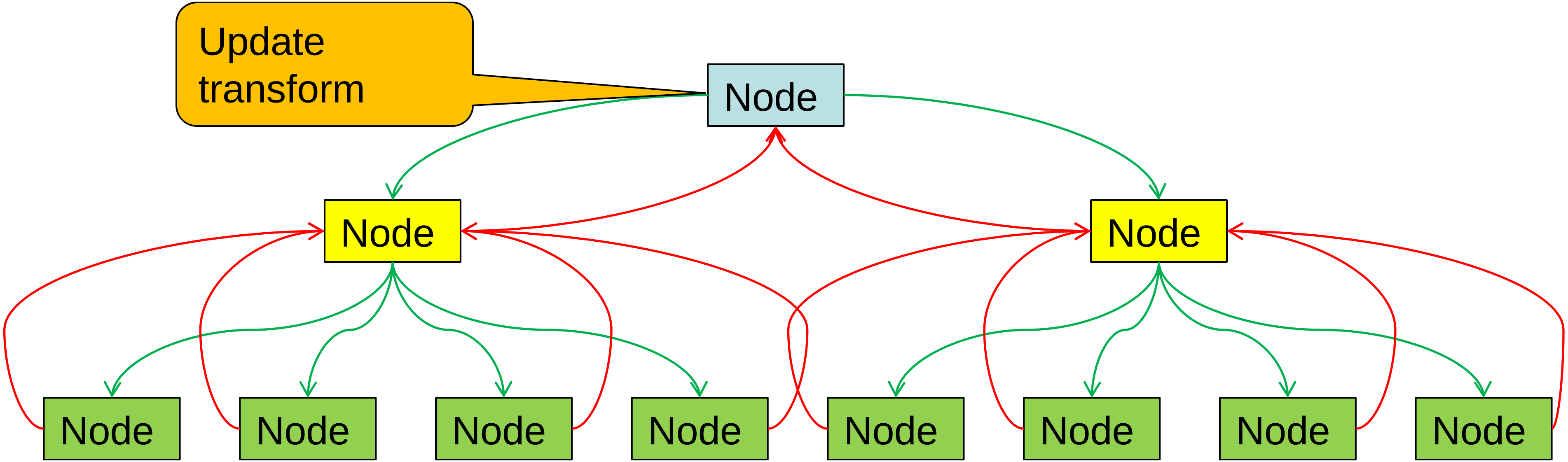


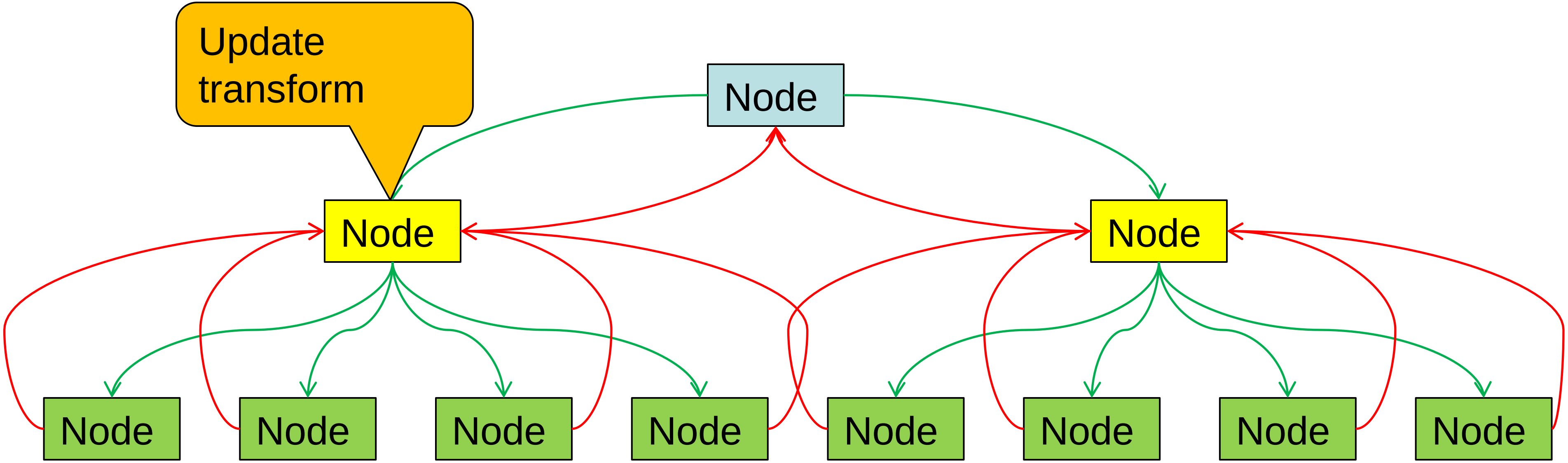
Ensure nodes and their data are contiguous in memory

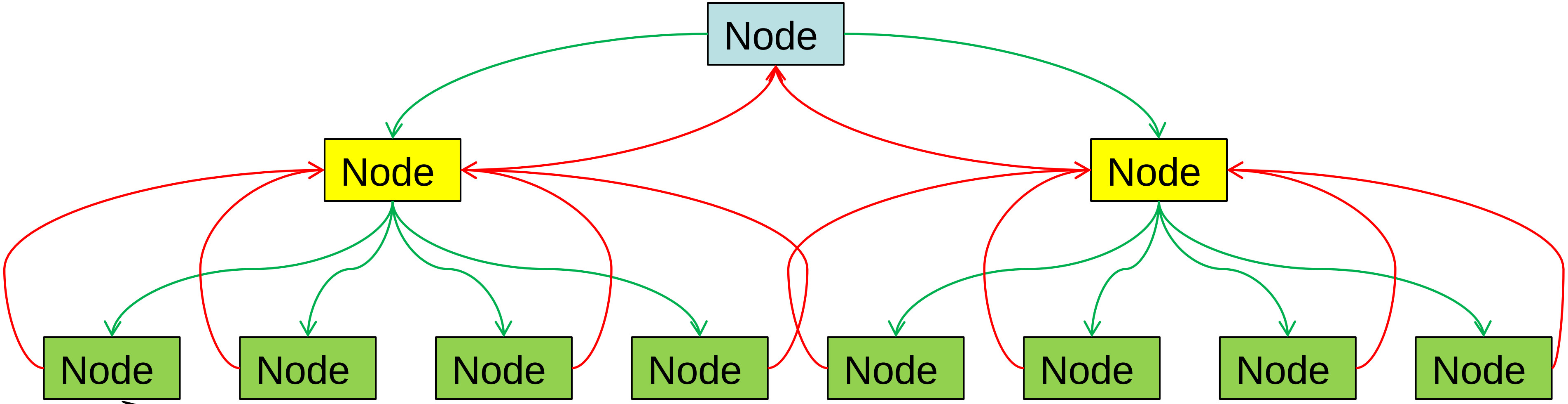
- Make the processing global rather than local
  - Pull the updates out of the objects.
    - No more virtuals
  - Easier to understand too – all code in one place.

Need to change some things…

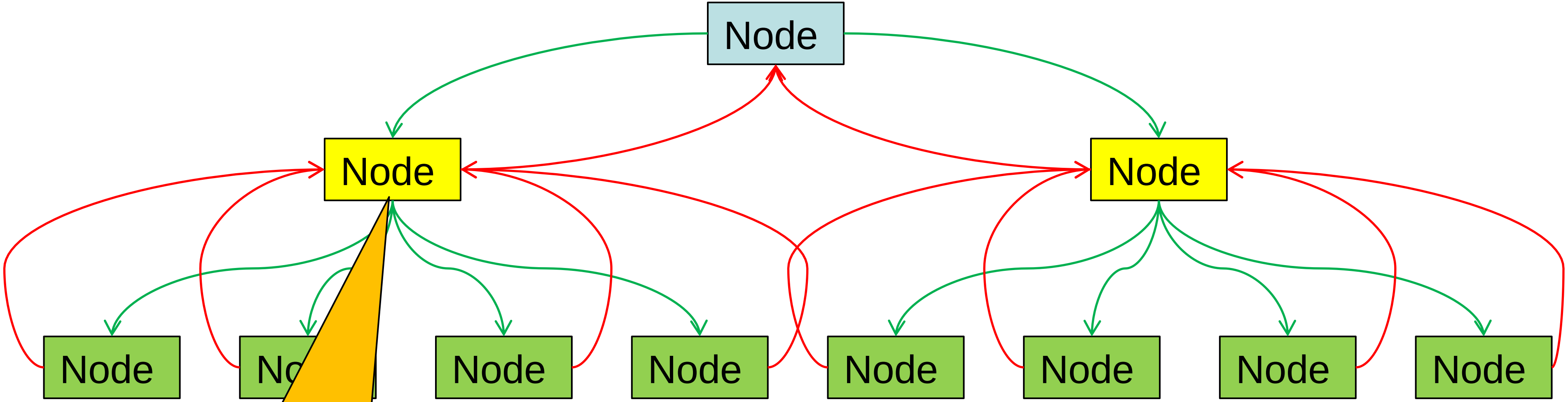
- OO version
  - Update transform top down and expand WBS bottom up


Update transform
Node
Node
Node
Node
Node
Node
Node
Node
Node
Node
Node

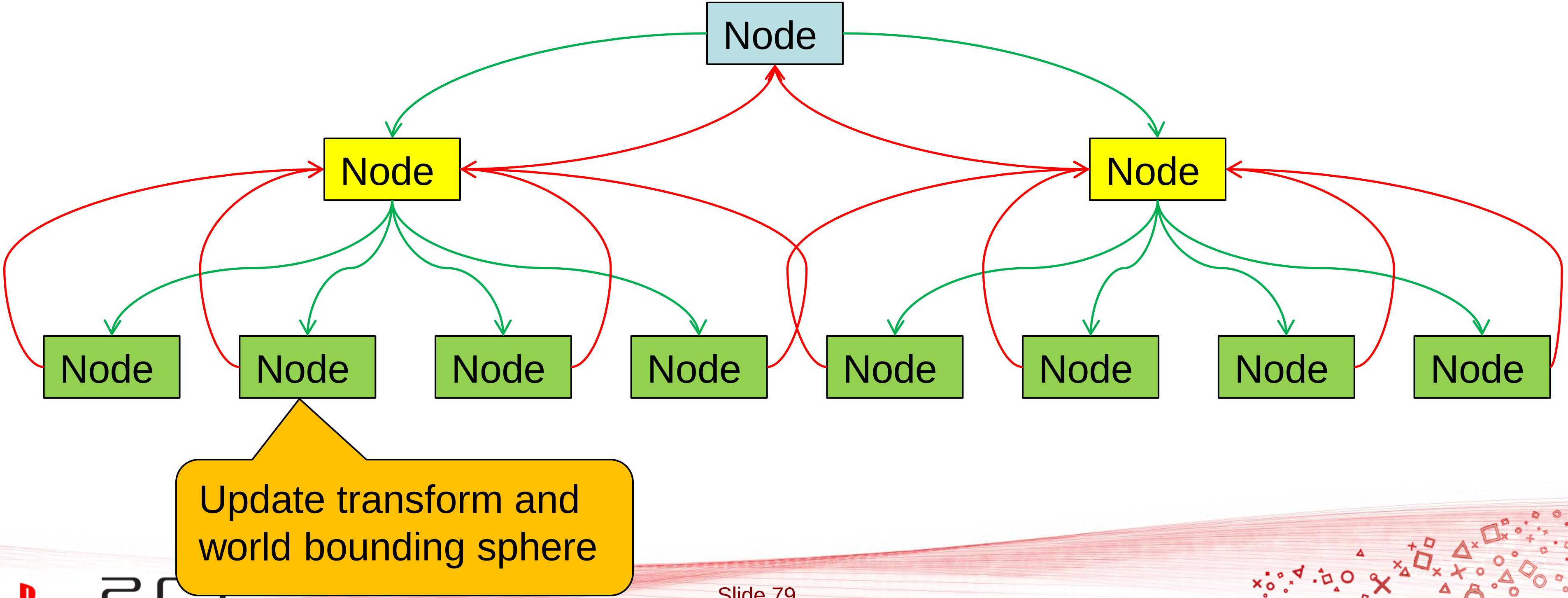

Update transform
Node
Node
Node
Node
Node
Node
Node
Node
Node
Node
Node

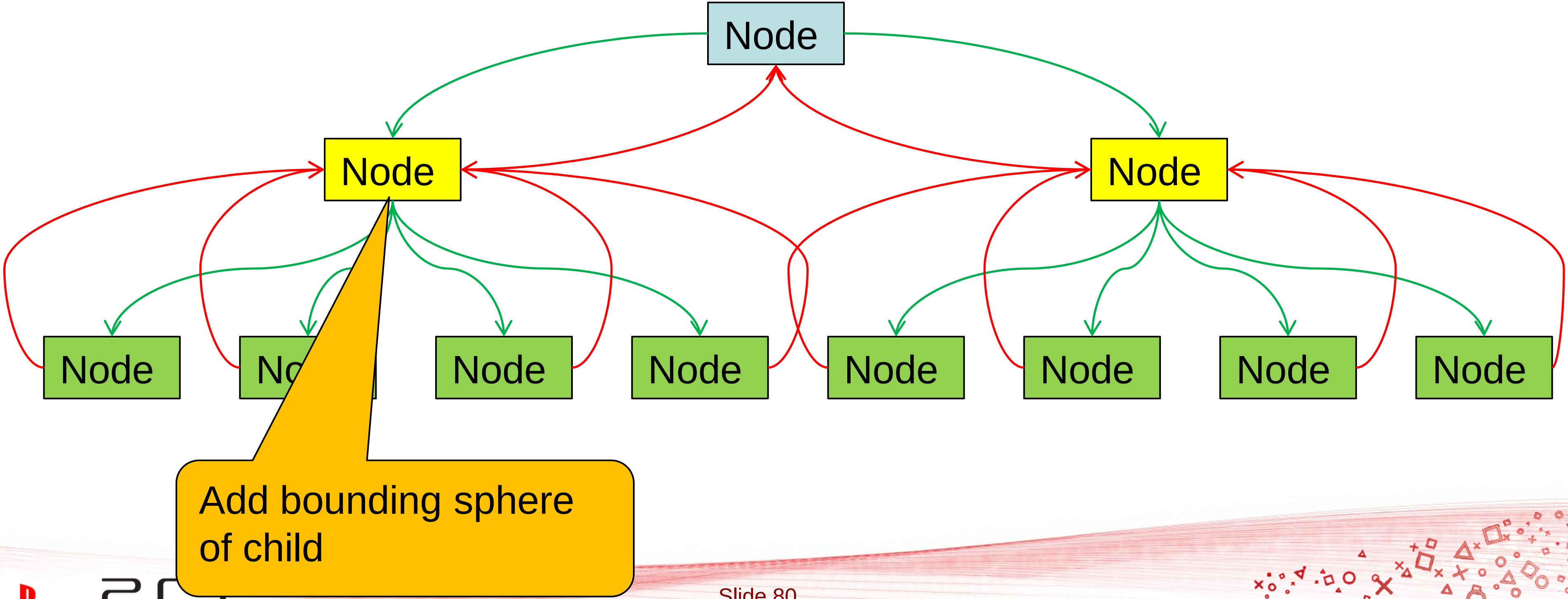


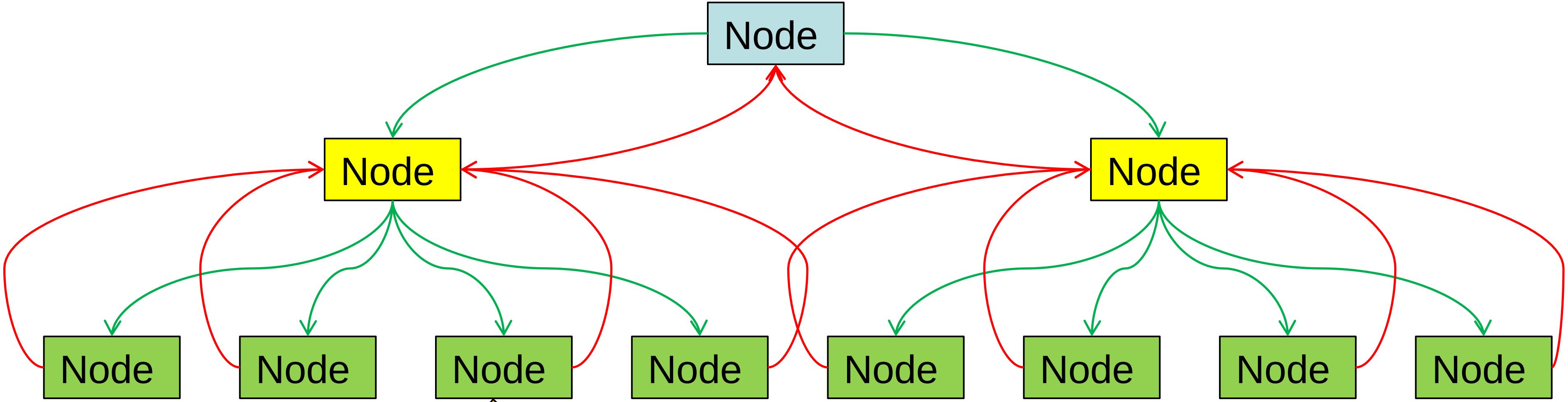
Update transform and world bounding sphere



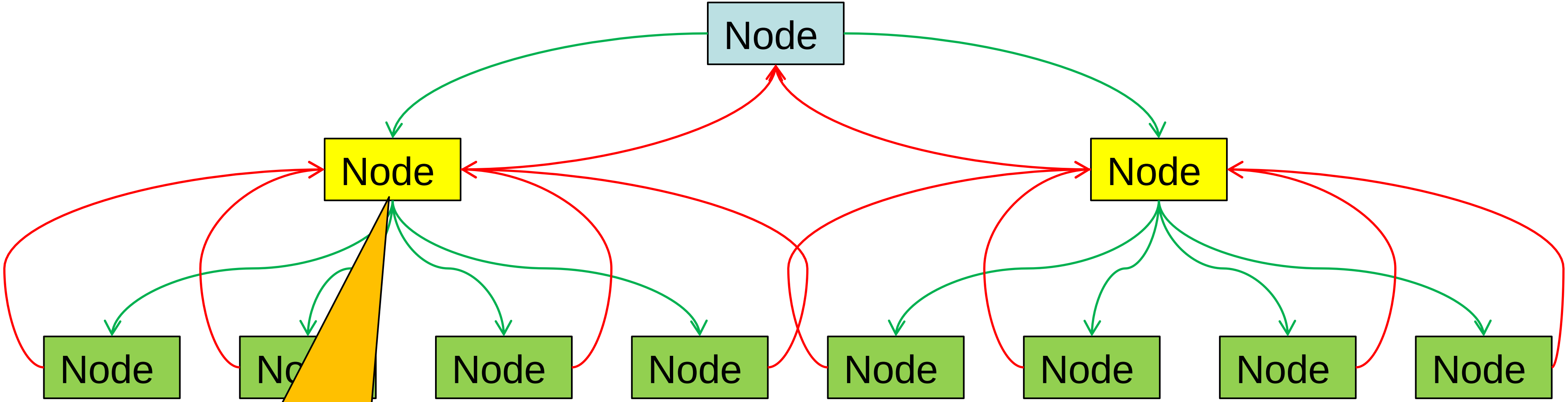
Add bounding sphere of child


Node
Node
Node
Node
Node
Node
Node
Node
Node
Node
Node
Update transform and world bounding sphere


Node
Node
Node
Node
Nc
Node
Node
Node
Node
Node
Node
Add bounding sphere of child



Update transform and world bounding sphere



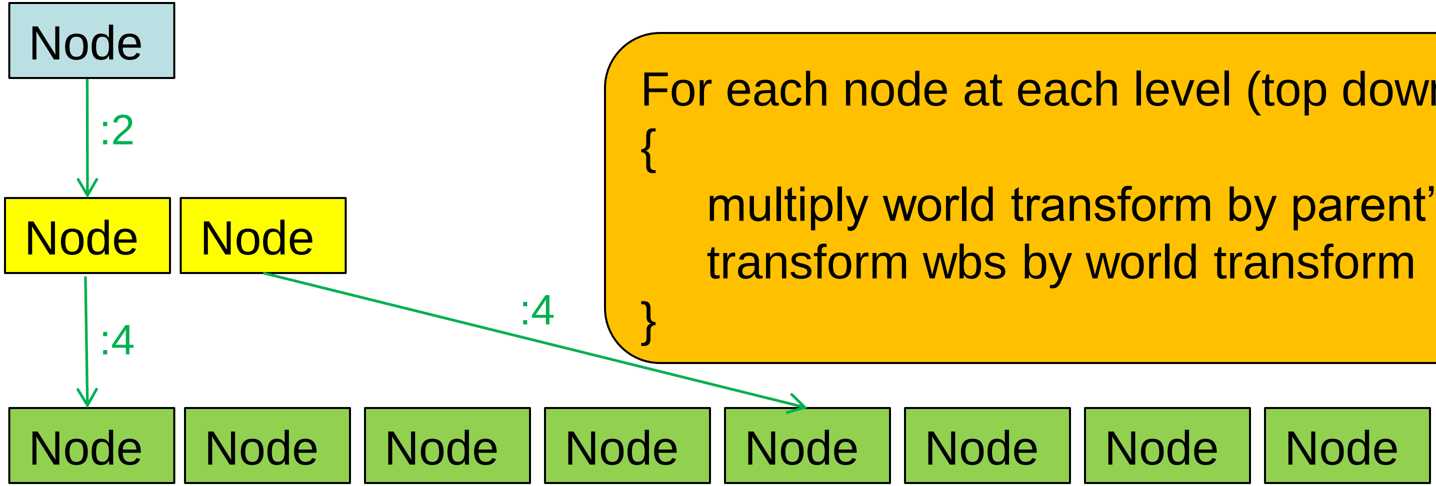
Add bounding sphere of child

- Hierarchical bounding spheres pass info up
- Transforms cascade down
- Data use and code is ‘striped’.
  - Processing is alternating

Conversion to linear

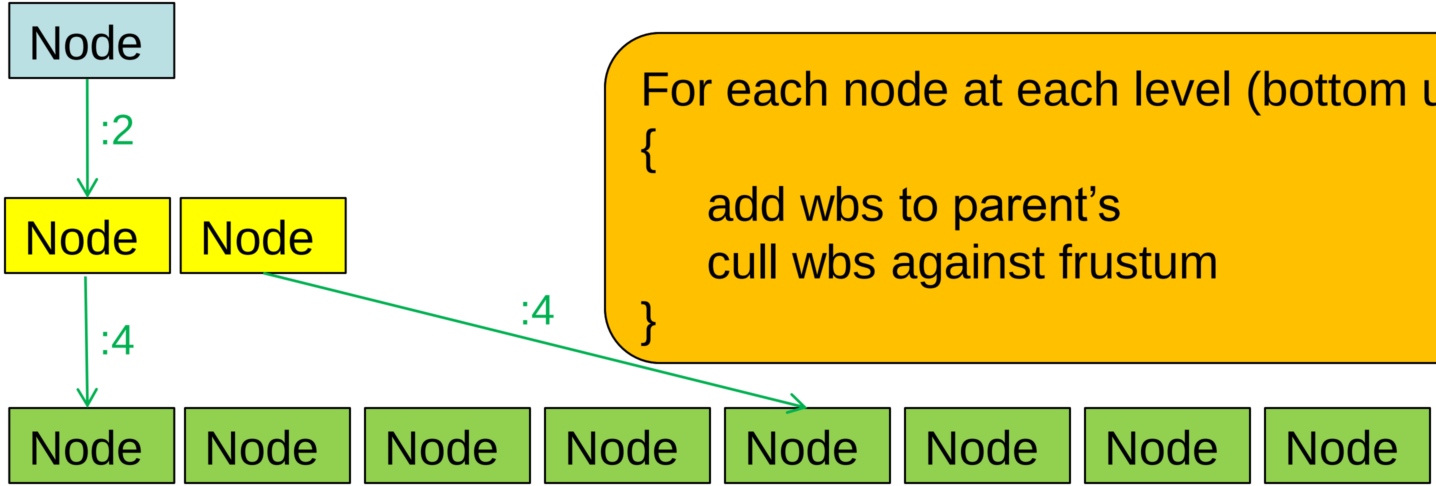
- To do this with a ‘flat’ hierarchy, break it into 2 passes
  - Update the transforms and bounding spheres(from top down)
  - Expand bounding spheres (bottom up)

Transform and BS updates




```
For each node at each level (top down)
{
    multiply world transform by parent's
    transform wbs by world transform
}
```

Update bounding sphere hierarchies



For each node at each level (bottom up)
{
    add wbs to parent's
    cull wbs against frustum
}

Update Transform and Bounding Sphere

How many children nodes to process

```
for(int j=0;j<size;j++)
{
    const int innerSize = parent->m_Objects.size();
    const Matrix4 *parentTransform = parent->m_WorldTransform;

    j+=innerSize;
    for(int k=0;k<innerSize;k++, wmat++, mat++, bs++, wbs++)
    {
        *wmat = (*parentTransform)*(*mat);
        *wbs = bs->Transform(wmat);
    }
    parent++;
}
```

Update Transform and Bounding Sphere

```
for(int j=0;j<size;j++)
{
    const int innerSize = parent->m_Objects.size();
    const Matrix4 *parentTransform = parent->m_WorldTransform;

    j+=innerSize;
    for(int k=0;k<innerSize;k++, wmat++, mat++, bs++, wbs++)
    {
        *wmat = (*parentTransform)*(*mat);
        *wbs = bs->Transform(wmat);
    }
    parent++;
}
```

For each child, update transform and bounding sphere

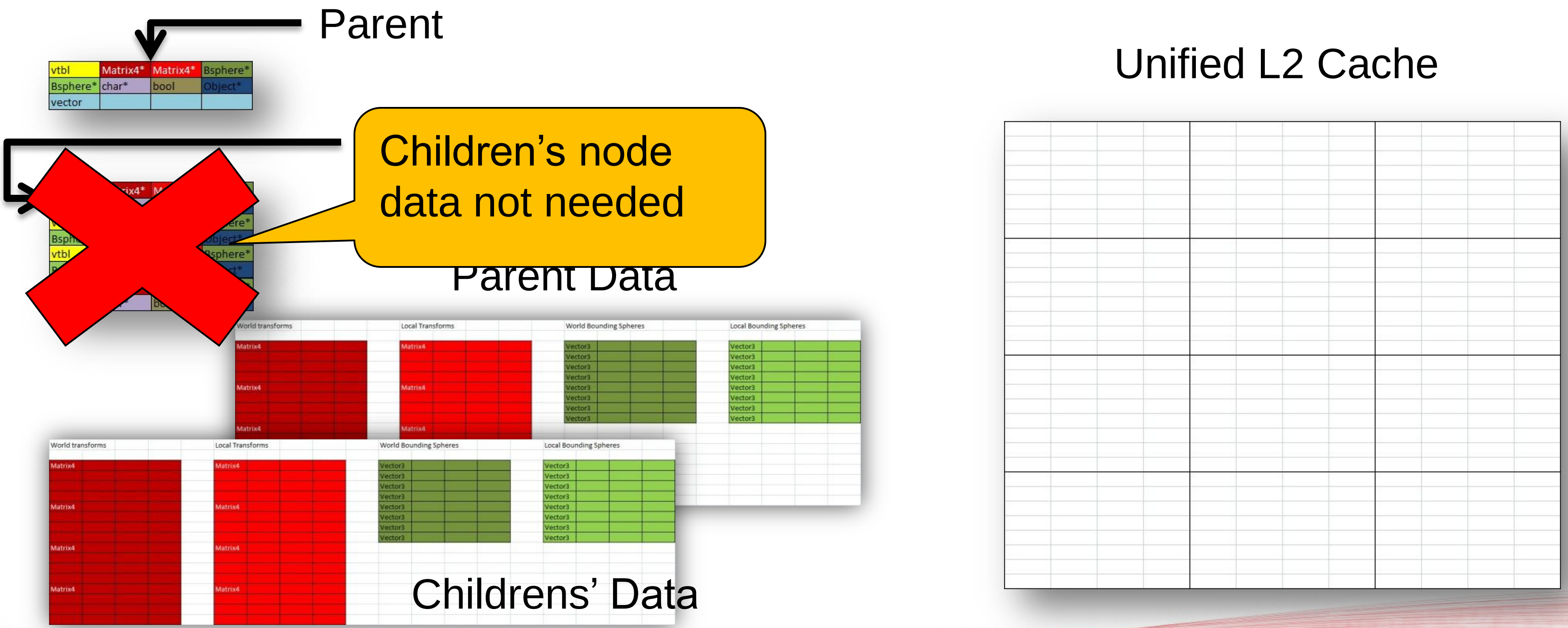
Update Transform and Bounding Sphere

```
for(int j=0;j<size;j++)
{
    const int innerSize = parent->m_Objects.size
    const Matrix4 *parentTransform = parent->m_WorldT

    j+=innerSize;
    for(int k=0;k<innerSize;k++, wmat++, mat++, bs++, wbs++)
    {
        *wmat = (*parentTransform)*(*mat);
        *wbs = bs->Transform(wmat);
    }
    parent++;
}
```

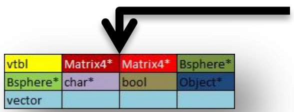
Note the contiguous arrays

So, what's happening in the cache?



Load parent and its transform

Parent




```
const int innerSize = parent->m_Objects.size();
const Matrix4 *parentTransform = parent->m_WorldTransform;
```

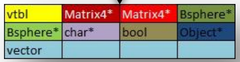
Parent Data

Childrens' Data

Unified L2 Cache

Load child transform and set world transform

Parent




```
for(int k=0;k<innerSize;k++, wmat++, mat++, bs++, wbs++)
{
    *wmat = (*parentTransform)*(*mat);
    *wbs = bs->Transform(wmat);
}
```

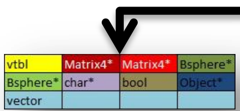
Parent Data

Childrens' Data

Unified L2 Cache

Load child BS and set WBS

Parent




```
for(int k=0;k<innerSize;k++, wmat++, mat++, bs++, wbs++)
{
    *wmat = (*parentTransform)*(*mat);
    *wbs = bs->Transform(wmat);
}
```

Unified L2 Cache

Parent Data

Childrens' Data

Load child BS and set WBS

Pare

vtbl	Matrix4*	Matrix4*	Bsphere*
Bsphere*	char*	bool	Object*
vector			

Next child is calculated with no extra cache misses !

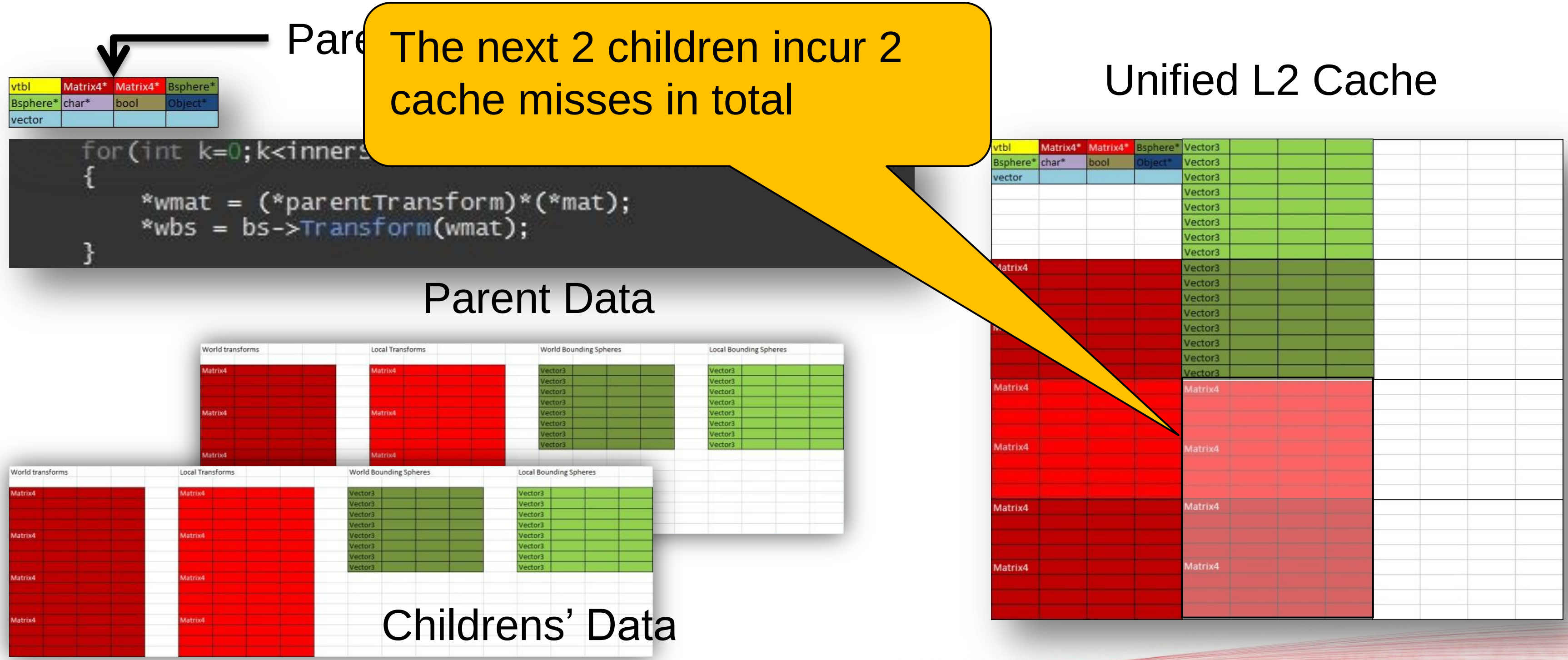
```
for(int k=0;k<inners
{
    *wmat = (*parentTransform)*(*mat);
    *wbs = bs->Transform(wmat);
}
```

Unified L2 Cache

Parent Data

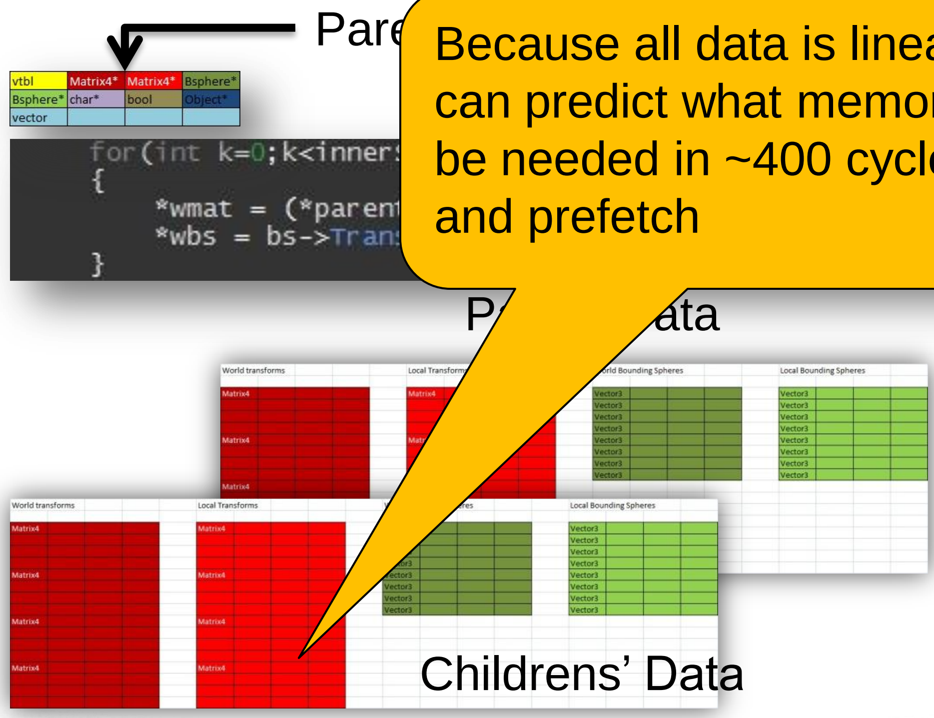
Childrens' Data

Load child BS and set WBS




```
for(int k=0;k<innerS
{
    *wmat = (*parentTransform)*(*mat);
    *wbs = bs->Transform(wmat);
}
```

Prefetching

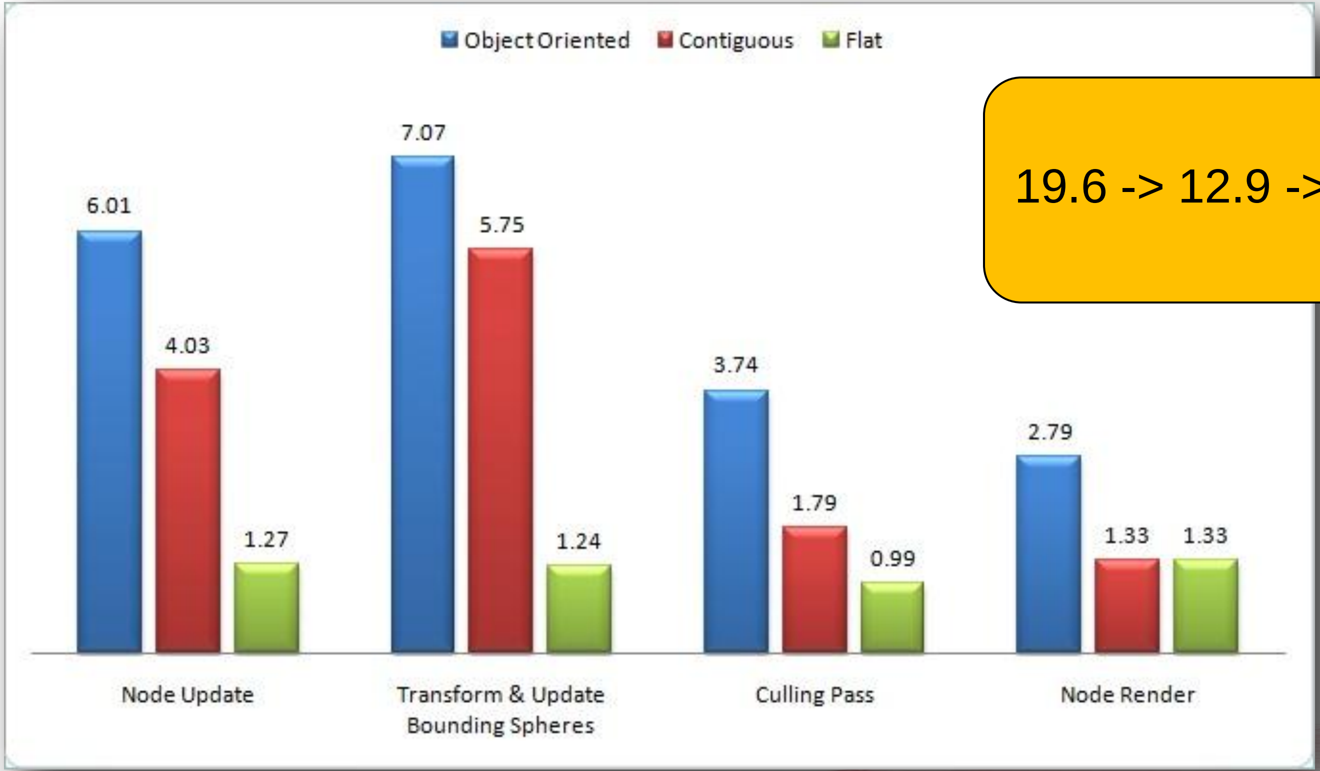


Because all data is linear, we can predict what memory will be needed in ~400 cycles and prefetch

Unified L2 Cache

- Tuner scans show about 1.7 cache misses per node.
- But, these misses are much more frequent
  - Code/cache miss/cache miss/code
  - Less stalling

Performance



19.6 -> 12.9 -> 4.8ms

Prefetching

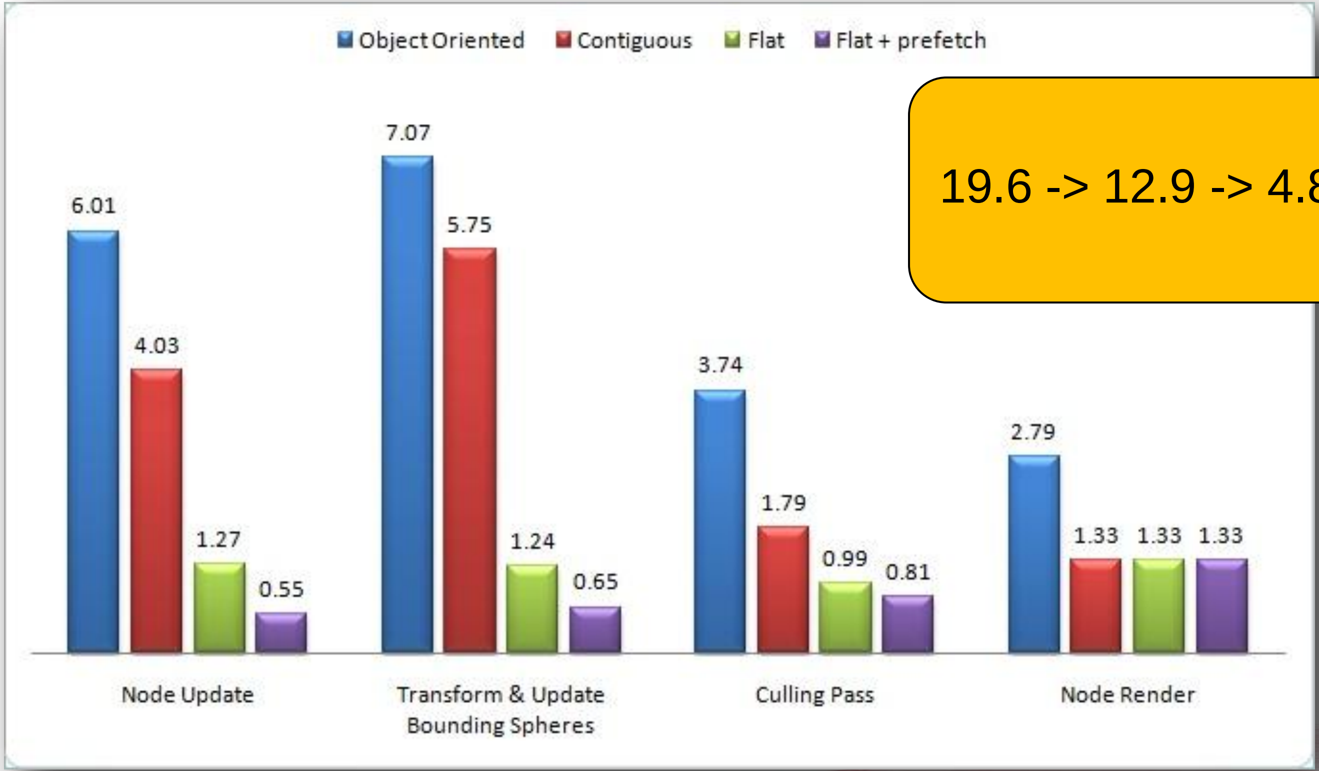
- Data accesses are now predictable
- Can use prefetch (dcbt) to warm the cache
  - Data streams can be tricky
  - Many reasons for stream termination
  - Easier to just use dcbt blindly
    - (look ahead x number of iterations)

Prefetching example

- Prefetch a predetermined number of iterations ahead
- Ignore incorrect prefetches

```
for(int i=0;i<m_CurrentNumTransforms;i++)
{
    __dcbt(m_Transforms[i+20]);
    *m_Transforms[i] = (*m_Transforms[i])*(*m_Transform);
}
```

Performance



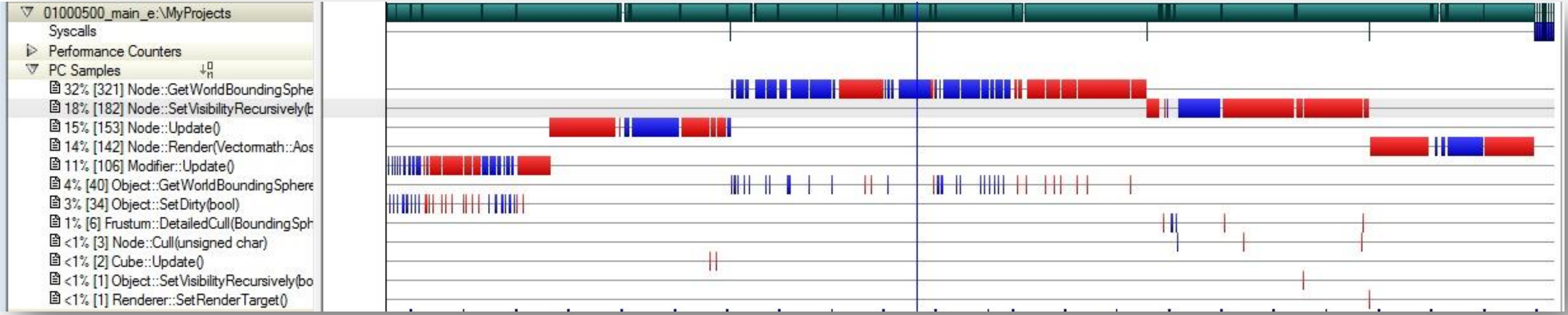
19.6 -> 12.9 -> 4.8 -> 3.3ms

A Warning on Prefetching

- This example makes very heavy use of the cache
- This can affect other threads' use of the cache
  - Multiple threads with heavy cache use may thrash the cache

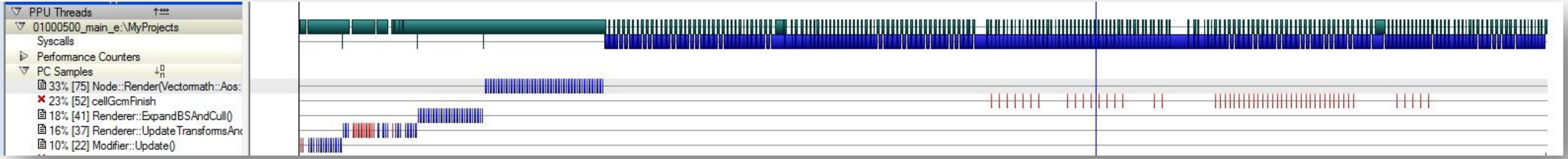
The old scan

~22ms

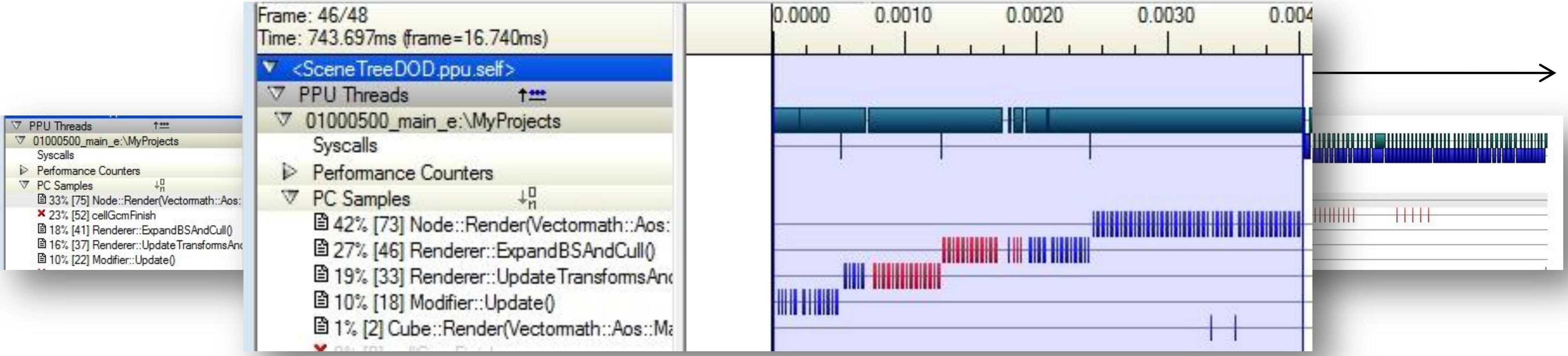


The new scan

~16.6ms



Up close



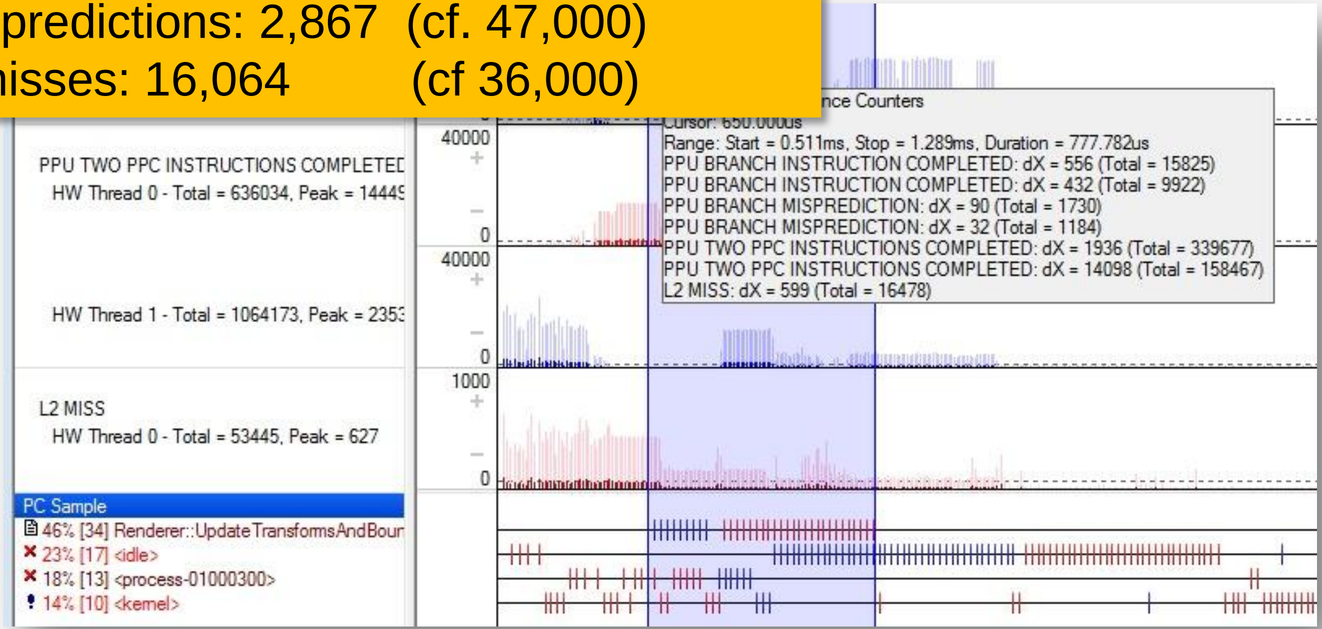
PPU Threads
01000500_main_e:\MyProjects
Syscalls
Performance Counters
PC Samples
33% [75] Node::Render(Vectormath::Aos:
23% [52] cellGcmFinish
18% [41] Renderer::ExpandBSAndCull()
16% [37] Renderer::UpdateTransformsAnc
10% [22] Modifier::Update()
Frame: 46/48
Time: 743.697ms (frame=16.740ms)
<SceneTreeDOD.ppu.self>
PPU Threads
01000500_main_e:\MyProjects
Syscalls
Performance Counters
PC Samples
42% [73] Node::Render(Vectormath::Aos:
27% [46] Renderer::ExpandBSAndCull()
19% [33] Renderer::UpdateTransformsAnc
10% [18] Modifier::Update()
1% [2] Cube::Render(Vectormath::Aos::Ma
0.0000
0.0010
0.0020
0.0030
0.004

Looking at the code (samples)

```
  -   -
151   3         const Node* parent = (Node*)node->m_Parent;
  -   -         // iterate through all the matrices at this level, multiplying them by their parent
  8   -         for(int j=0;j<size;j++)//,node++, wmat++, mat++, wbs++, bs++)
  -   -         {
  -   -             const int innerSize = parent->m_Objects.size();
 25   -             const Matrix4 *parentTransform = parent->m_WorldTransform;
  -   -
  2   -             j+=innerSize;
372   9             for(int k=0;k<innerSize;k++, wmat++, mat++, bs++, wbs++, node++)
  -   -             {
233   2                 __dcbt(wmat+lookAhead);
146   3                 __dcbt(mat+lookAhead);
317   5                 __dcbt(bs+lookAhead);
117   4                 __dcbt(wbs+lookAhead);
  -   -                 *wmat = (*parentTransform)*(*mat);
  -   -  //             *wmat = (*node->m_Parent->m_WorldTransform)*(*mat);
  -   -
 32   1                 *wbs = bs->Transform(wmat);
  -   -             }
  -   -             parent++;
```

Performance counters

Branch mispredictions: 2,867 (cf. 47,000)
L2 cache misses: 16,064 (cf 36,000)



In Summary

- Just reorganising data locations was a win
- Data + code reorganisation= dramatic improvement.
- + prefetching equals even more WIN.

OO is not necessarily EVIL

- Be careful not to design yourself into a corner
- Consider data in your design
  - Can you decouple data from objects?
  - …code from objects?
- Be aware of what the compiler and HW are doing

Its all about the memory

- Optimise for data first, then code.
  – Memory access is probably going to be your biggest bottleneck
- Simplify systems
  – KISS
  – Easier to optimise, easier to parallelise

Homogeneity

- Keep code and data homogenous
  - Avoid introducing variations
  - Don’t test for exceptions – sort by them.
- Not everything needs to be an object
  - If you must have a pattern, then consider using Managers

Remember

- You are writing a GAME
  - You have control over the input data
  - Don’t be afraid to preformat it – drastically if need be.
- Design for specifics, not generics (generally).

Data Oriented Design Delivers

- Better performance
- Better realisation of code optimisations
- Often simpler code
- More parallelisable code

The END