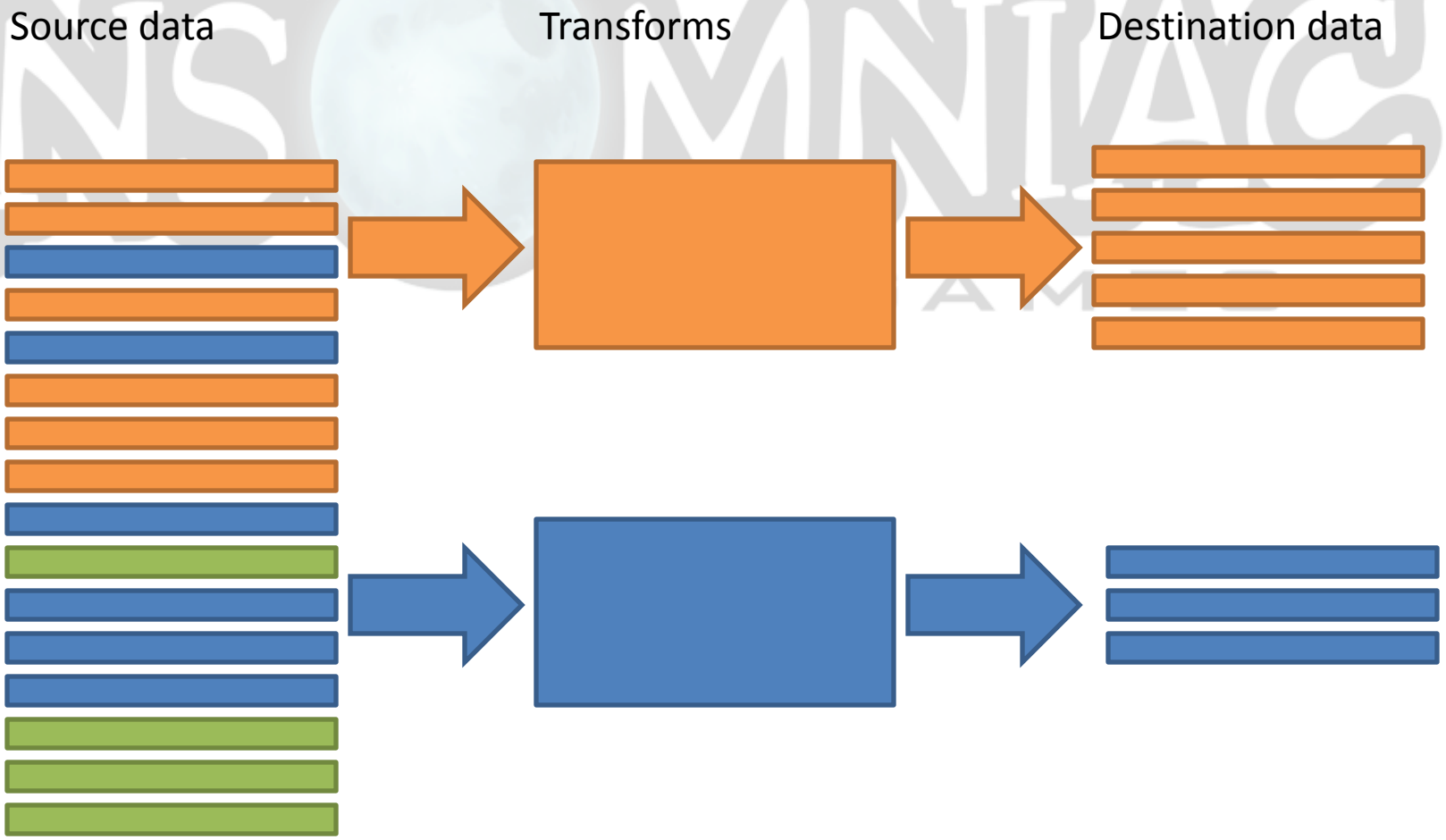
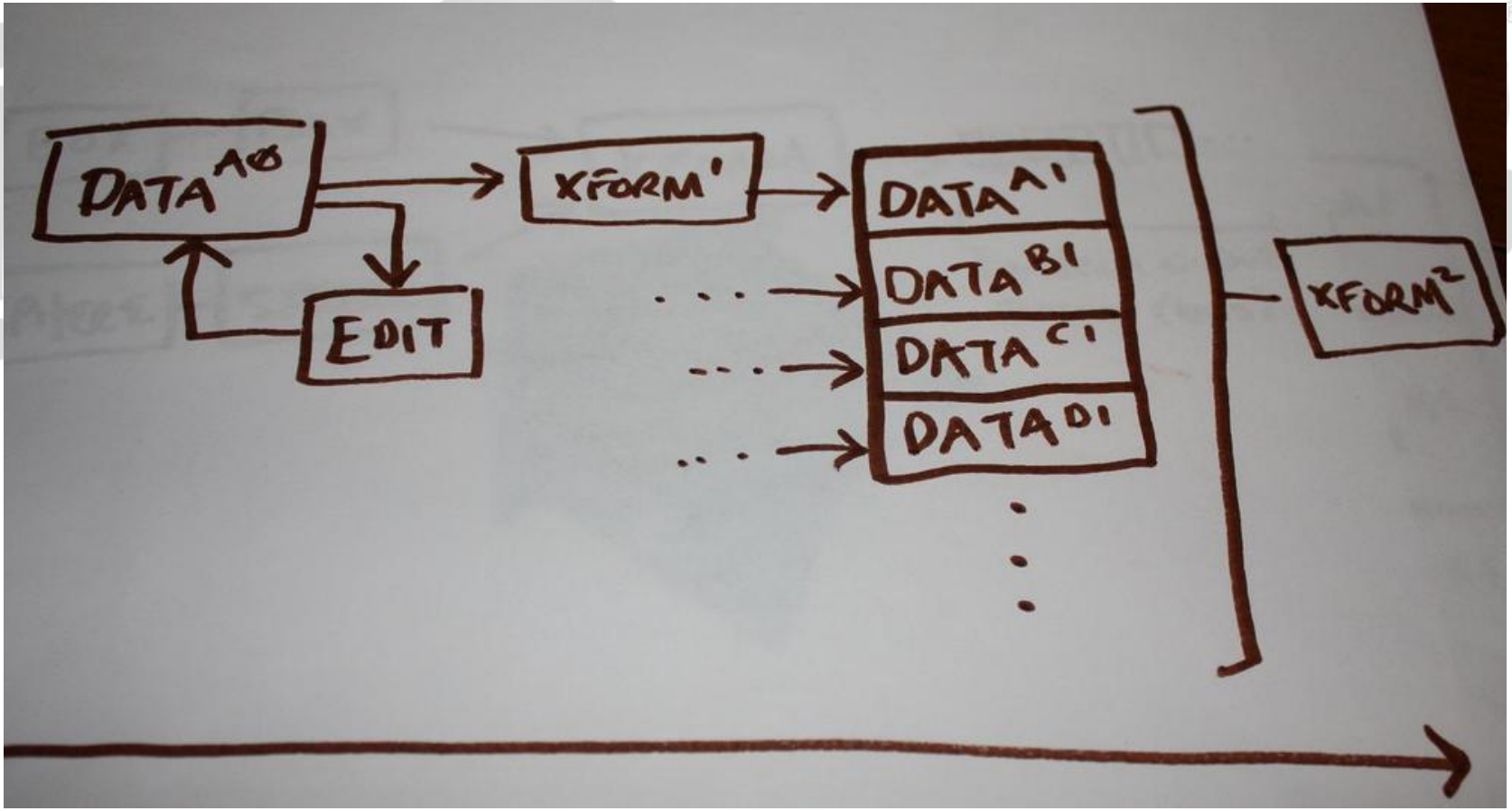
Data Design Class #1
Unrelated Transforms

Mike Acton 29 Mar 2010

What is an unrelated transform?



Solving related transforms is intuitive



Data design fundamental: **Two different problems imply two different data designs**.

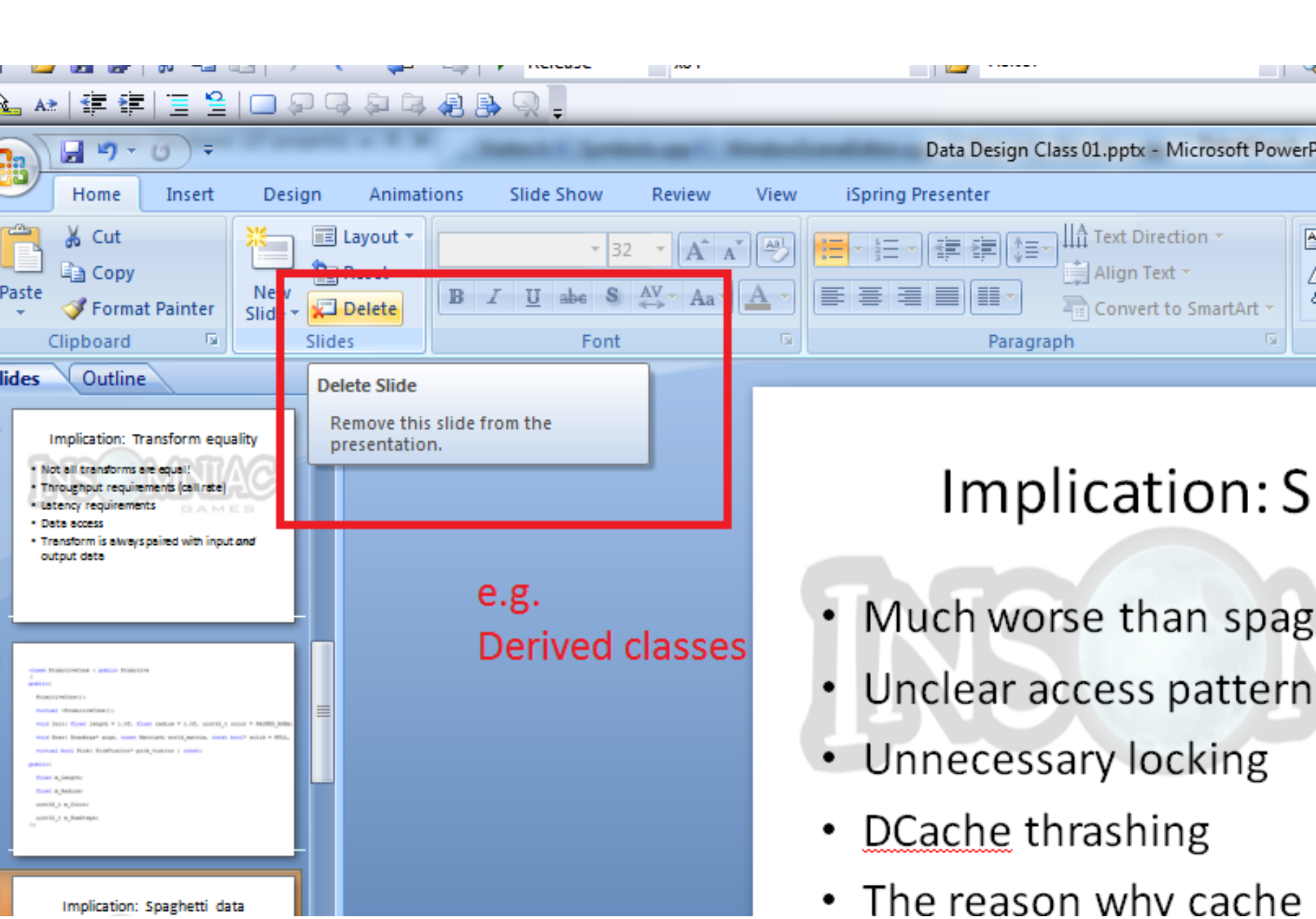
Implication: Transform equality

- Not all transforms are equal!
- Throughput requirements (call rate)
- Latency requirements
- Data access
- Transform is always paired with input ***and*** output data

```
class PrimitiveCone : public Primitive
{
public:

  PrimitiveCone();

  virtual ~PrimitiveCone();

  void Init( float length = 1.0f, float radius = 1.0f, uint32_t color = PACKED_RGBA(

  void Draw( DrawArgs* args, const Matrix4& world_matrix, const bool* solid = NULL, c

  virtual bool Pick( PickVisitor* pick_visitor ) const;

public:

  float m_Length;

  float m_Radius;

  uint32_t m_Color;

  uint32_t m_NumSteps;
};
```

Implication: Spaghetti data

- Much worse than spaghetti code!
- Unclear access patterns
- Unnecessary locking
- DCache thrashing
- The reason why cache sizes are bloated in h/w
- No statistical relationship


Data Design Class 01.pptx - Microsoft PowerP
Home
Insert
Design
Animations
Slide Show
Review
View
iSpring Presenter
Cut
Copy
Format Painter
Paste
Clipboard
Layout
Delete
Slides
Font
Text Direction
Align Text
Convert to SmartArt
Paragraph
Outline
Delete Slide
Remove this slide from the presentation.
Implication: Transform equality
Not all transforms are equal!
Throughput requirements (call rate)
Latency requirements
Data access
Transform is always paired with input and output data
Implication: Spaghetti data
e.g.
Derived classes
Implication: S
Much worse than spag
Unclear access pattern
Unnecessary locking
DCache thrashing
The reason why cache

Implication: Monolithic structs

- Poor memory characteristics
- Unknown usage patterns
- Transactions unclear

MobyClass

```
struct MobyClass
{
   vec4f                    m_bsphere;
   i16                      m_bind_pose_inv_offset;
   u16                      m_flags;
   i16                      m_runtime_enum;
   u16                      m_frag_cnt;
   u16                      m_segment_cnt;
   u16                      m_shader_set_cnt;
   u16                      m_joint_cnt;
   u16                      m_num_render_bspheres;
   u16                      m_main_heap_handle;
   u16                      m_vram_heap_handle;
   IGG::MobySegment*        m_segments;
   IGG::JointHierarchy*     m_joint_hierarchy;
   void*                    m_render_bspheres;

   COLL::Object*            m_coll_prim_proto;
   COLL::Object*            m_coll_tri_mesh_proto;
   u32                      m_index_data;
   u32                      m_vertex_data;

   // 64 bytes

   i32                      m_default_update_enum;
   f32                      m_default_draw_dist;
   f32                      m_default_update_dist;
   u16                      m_clip_data_cnt;
   u16                      m_anim_query_handle;

   u64                      m_anim_set_tuid;                // tuid
   IGG::MobyAnimSet*        m_anim_set;                     // point
   AnimGamePlayData*        m_anim_gp_settings;

   IGG::ComplexAnim::ClipData*      m_clip_data;
   IGG::DynamicJoint::Hierarchy*    m_dynamic_joint_hierarchy;
   IGG::MobySegCollPrimInfo*        m_seg_coll_prim_info;   // segme
   IGG::LooseAttSystemInfo*         m_loose_att_data;       // loose

   f32                      m_vertex_scale;
   f32                      m_tex_stream_dist;
   u16                      m_shadow_merge_groups;
   u16                      m_pad0;
   f32                      m_shadow_aabb_ext;

   // 128 bytes

   PHYSICS::PhysicsClass*   m_physics;
   PhysicsInfoDat*          m_physics_info;
   PhysicsJointDat*         m_physics_dat;
   BangleJointDat*          m_bangle_joint_info;
   u32                      m_bangle_cnt;
   u8*                      m_bangle_segment_ids;
   u64                      m_bangle_default_draw;

   u32                      m_phase_joint_group_cnt;
   u16*                     m_phase_joint_indices_per_group_cnt;
   u16**                    m_phase_joint_indices;
   u8*                      m_particle_def;

   u64                      m_tuid;                         // tuid
   char*                    m_name;                         // name
   BangleGeomSimDat*        m_bangle_geom_sim_dat;

   // 192 bytes

   u64                      m_bangle_cheap_chunk;
   NavEffectorDat*          m_nav_eff_dat;
   NavClueDat*              m_nav_clue_dat;

   IGG::MorphInfo*             m_morph_info;
   u32                         m_loose_att_data_size; //ask reddy i
   void*                       m_r2o_interact_data;
   PHYSICS::DestructionClass *m_destruction;
   u32                         m_pad[8];

   // 256 bytes
};
```

Note: Compare components

Implication: Classification flags

- Need to distinguish cases
- A lot of additional implicit state machines
- Unnecessary checks/tests
- Additional points of failure
- Nonsense combinations

```
29
30  struct VisitorState
31  {
32      Math::Matrix4 m_Matrix;
33      bool m_Highlighted;
34      bool m_Selected;
35      bool m_Live;
36      bool m_Selectable;
37
38      void Init( const Math::Matrix4 matrix
39
40      // Deprecated
41      VisitorState();
42      VisitorState( const Math::Matrix4& ma
43      VisitorState( const Math::Matrix4& ma
44  };
45
```

```
23
24    struct Grid
25    {
26    public:
27      uint32_t m_Width;
28      uint32_t m_Length;
29      float m_MajorStep;
30      float m_MinorStep;
31
32    private:
33      GridTypes::Enum   m_GridType;
34      Matrix3           m_GridRotation;
35
36      [ ... ]
37
38      void Draw( RenderLayerBuffer* layer, uint32_t layer_id, bool
39      void Draw( RenderLayerBuffer* layer, uint32_t layer_id, cons
40
41      bool IsMultiple( float value, float multiple );
42      uint32_t RenderWidthLines( Vector3* lines, float length, boo
43      uint32_t RenderLengthLines( Vector3* lines, float width, boo
44      uint32_t RenderOriginLines( Vector3* lines, float length, fl
45      float GetGridRenderAdjustmentLength( const Vector3& offset )
46      float GetGridRenderAdjustmentWidth( const Vector3& offset );
47    };
48  }
```

Implication: Unnecessary virtuals

- Asking the wrong question:
- Don’t ask: “What does this do?”
- Ask: “What am I ***doing*** with this data?”

```
270
271   class FrustumPickVisitor : virtual public PickVisitor
272   {
273   public:
274
275     [ ... ]
276
277     virtual void Transform();
278     virtual bool PickPoint( const Math::Vector3& p, float err = Math:
279     virtual bool PickSegment( const Math::Vector3& p1, const Math::Ve
280     virtual bool PickTriangle( const Math::Vector3& v0, const Math::V
281     virtual bool PickSphere( const Math::Vector3& center, float radiu
282     virtual bool InstersectsSphere( const Math::Vector3& center, floa
283     virtual bool PickBox( const Math::AlignedBox& box );
284     virtual bool IntersectsBox( const Math::AlignedBox& box );
285
286   protected:
287     Math::Frustum m_PickSpaceFrustum;
288     Math::Frustum m_WorldSpaceFrustum;
289
290     bool AddHitPoint( const Math::Vector3& p );
291     bool AddHitSegment( const Math::Vector3& p1, const Math::Vector3&
292     bool AddHitTriangle( const Math::Vector3& v0, const Math::Vector3
293     bool AddHitSphere( const Math::Vector3& center );
294     bool AddHitBox( const Math::AlignedBox& box );
295   };
```

Implication: Unoptimizable

- Obvious: Global optimizations more valuable than local ones.
- Data not organized on global access patterns can’t be globally optimized.

Implication: Poor concurrency

- Short answer: Can’t.
- Typical solution must enforce sequence.
  - i.e. Over-locking

Implication: Code > Data

- Bears repeating:
  - We do not write code.
  - We transform data.
  - Code is a tool to do that.

Discussion Topic

"Visitor Pattern"