



# Designing and Evaluating Reusable Components

Casey Muratori

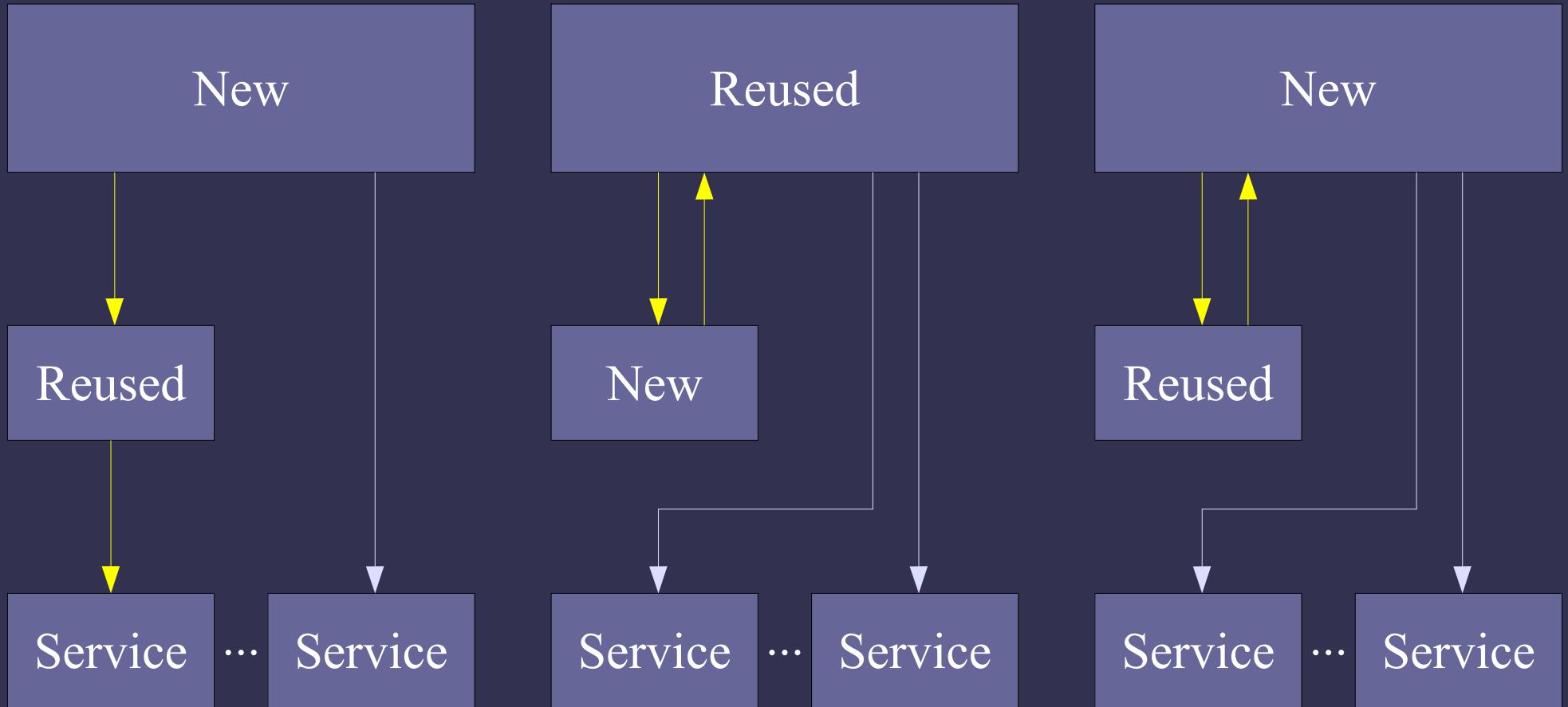
[casey@mollyrocket.com](mailto:casey@mollyrocket.com)

Code reuse. Sigh.

Layer

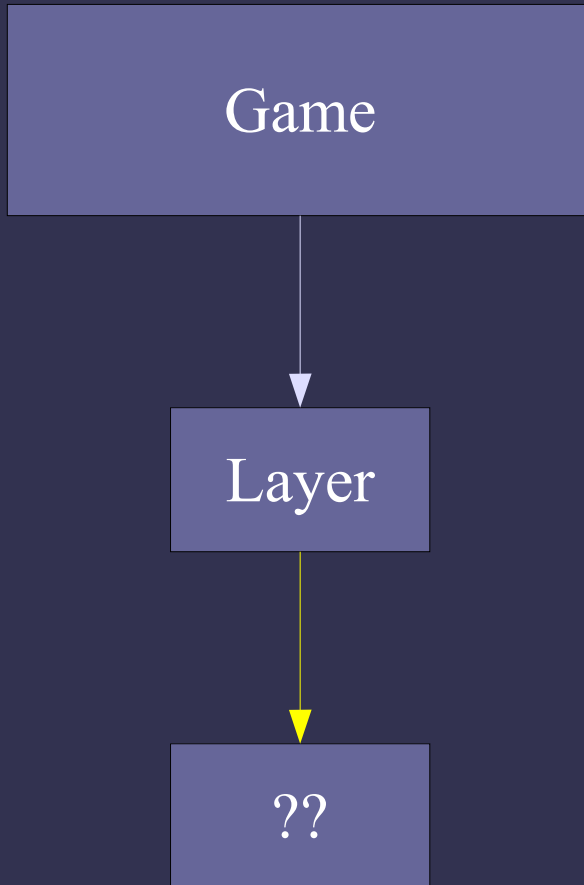
Engine

Component

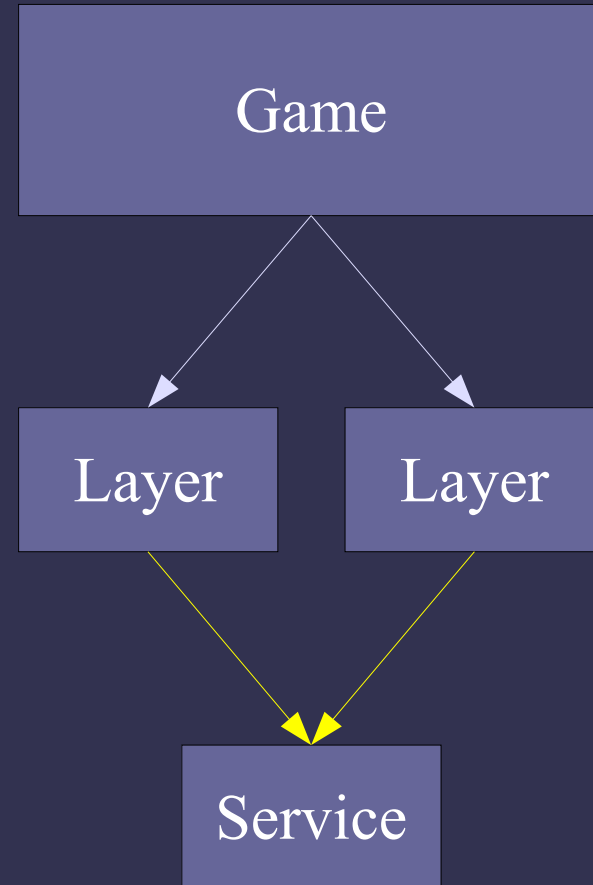


Three types of reuse

Layers require services  
(standards)

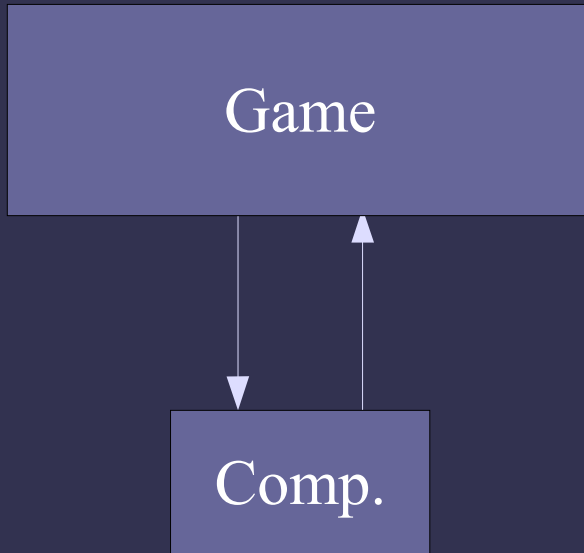


Layers can conflict

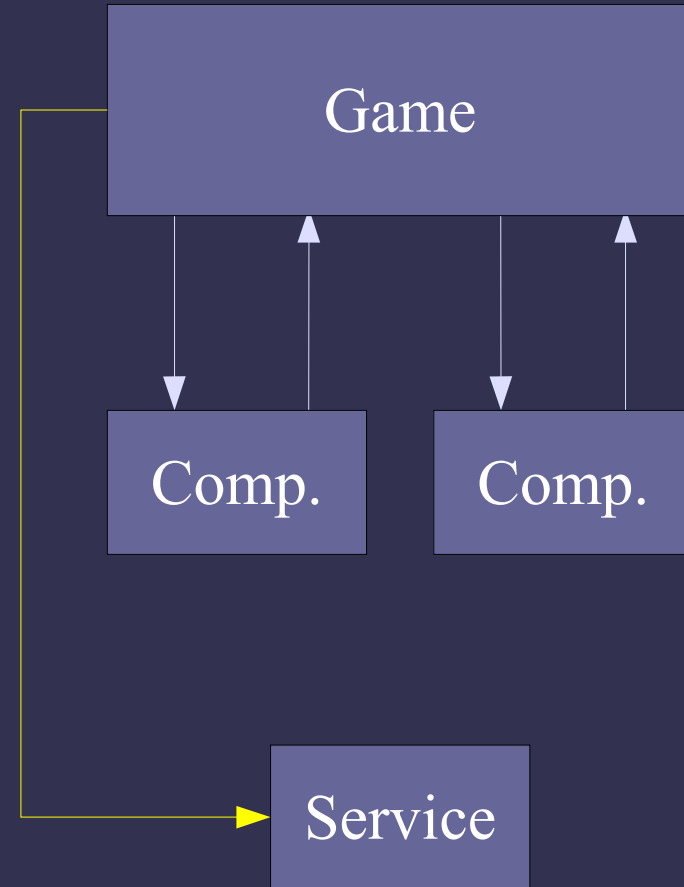


Layers alone are insufficient

Service not required

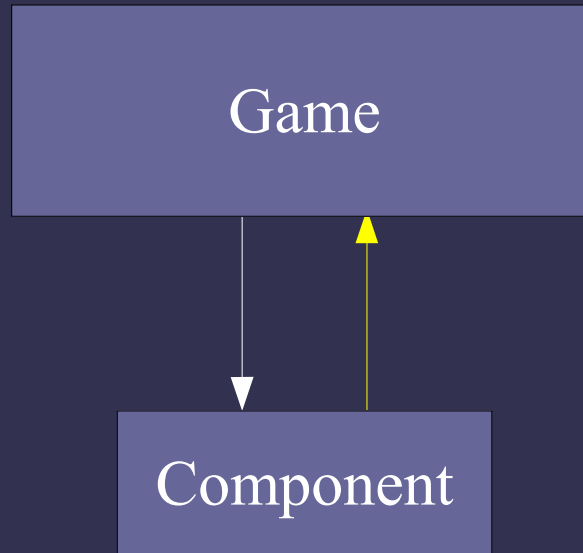


Conflicts resolved  
in game

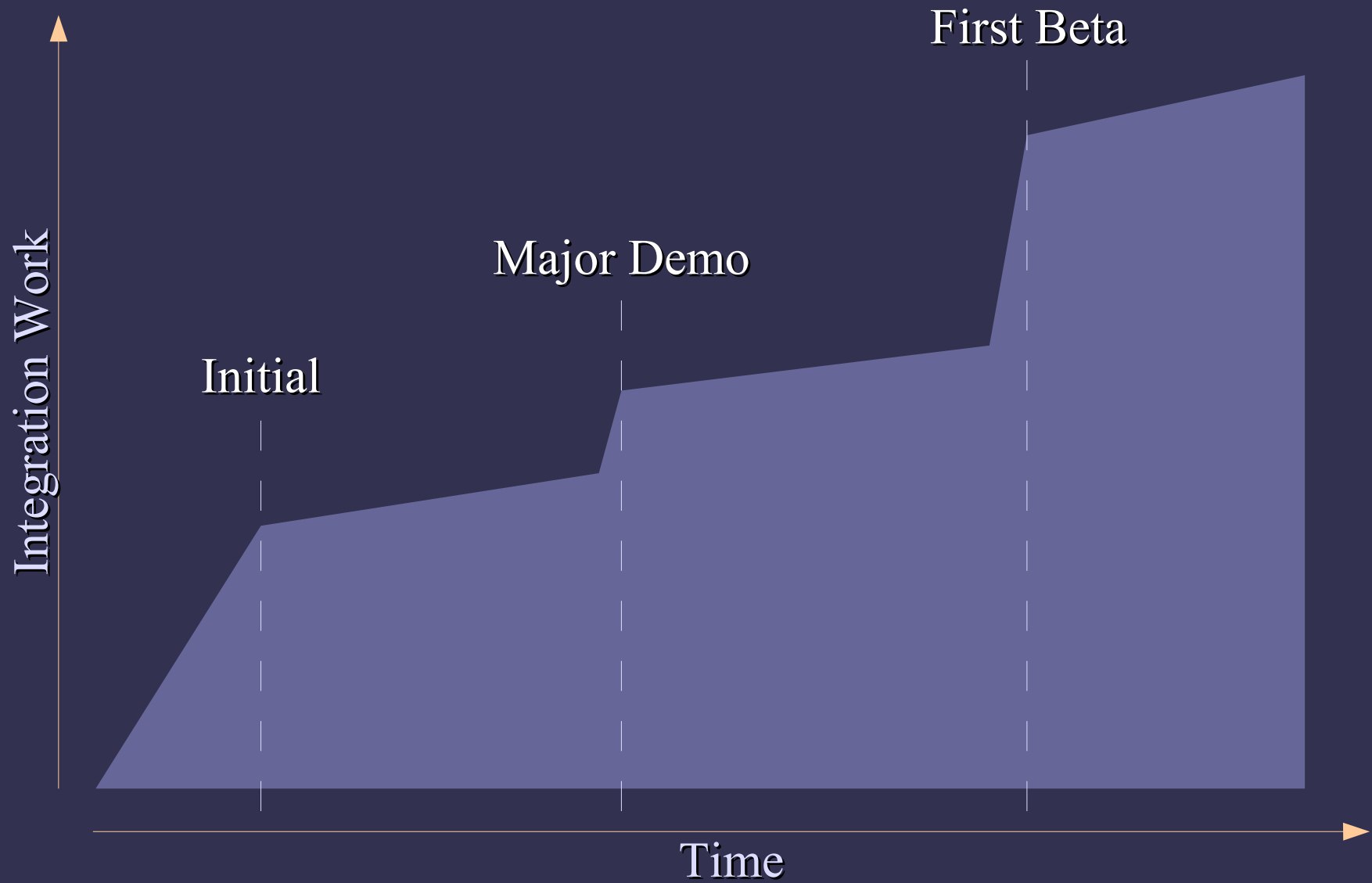


Components provide solution

Components solve most remaining reuse problems,  
but are significantly harder to design.



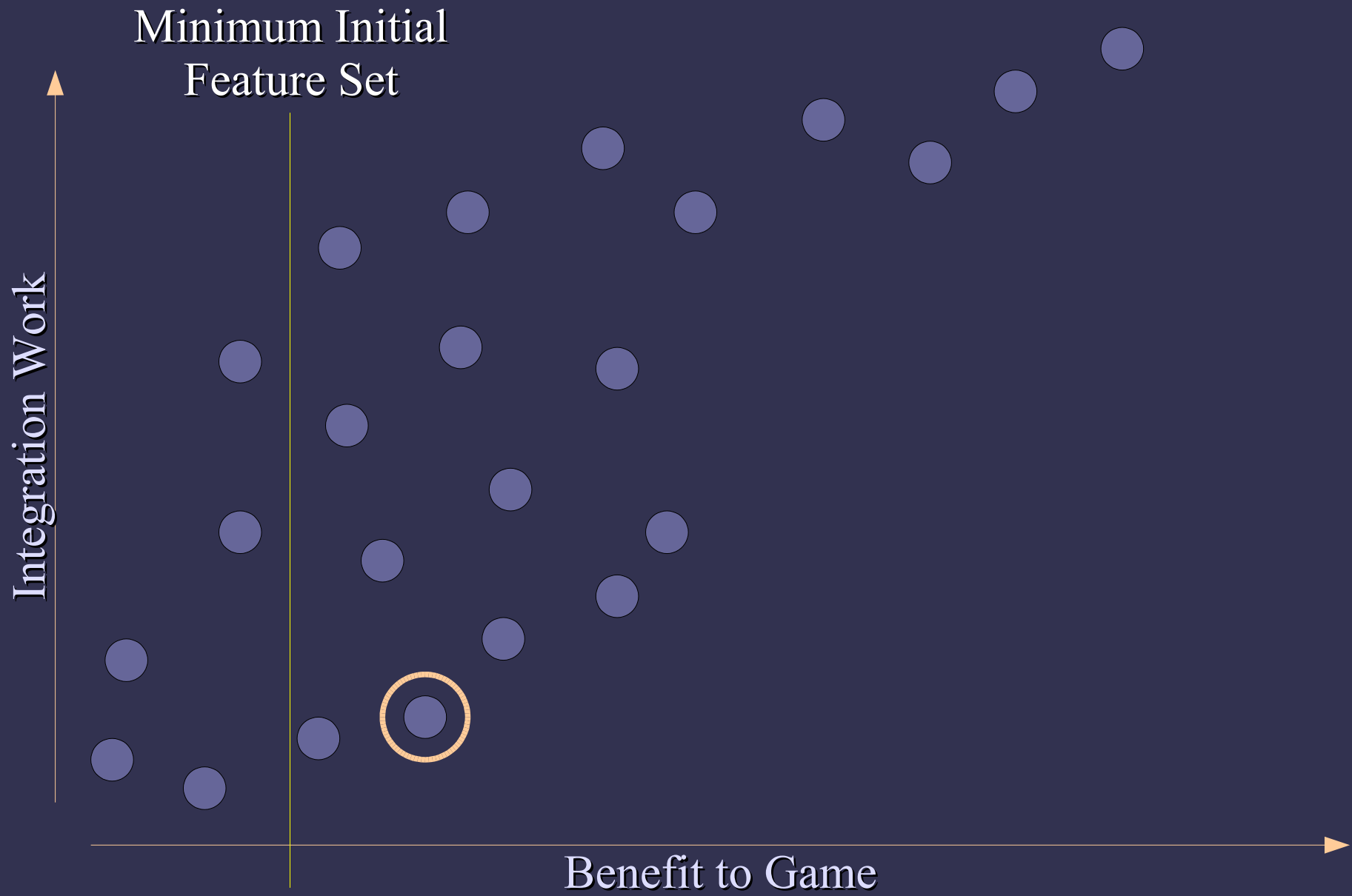
Components are integral



Integration grows in spurts

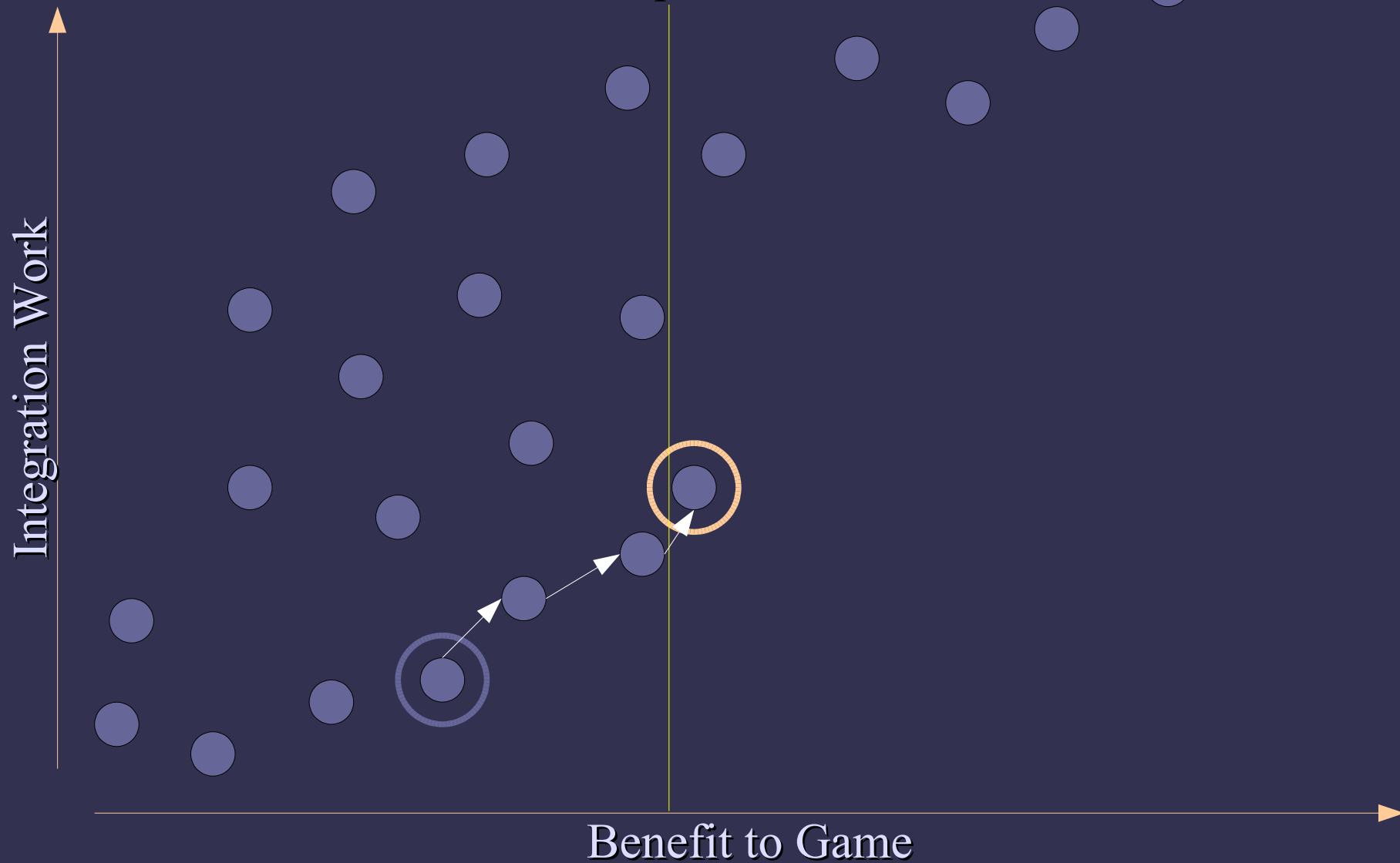


# Integration options

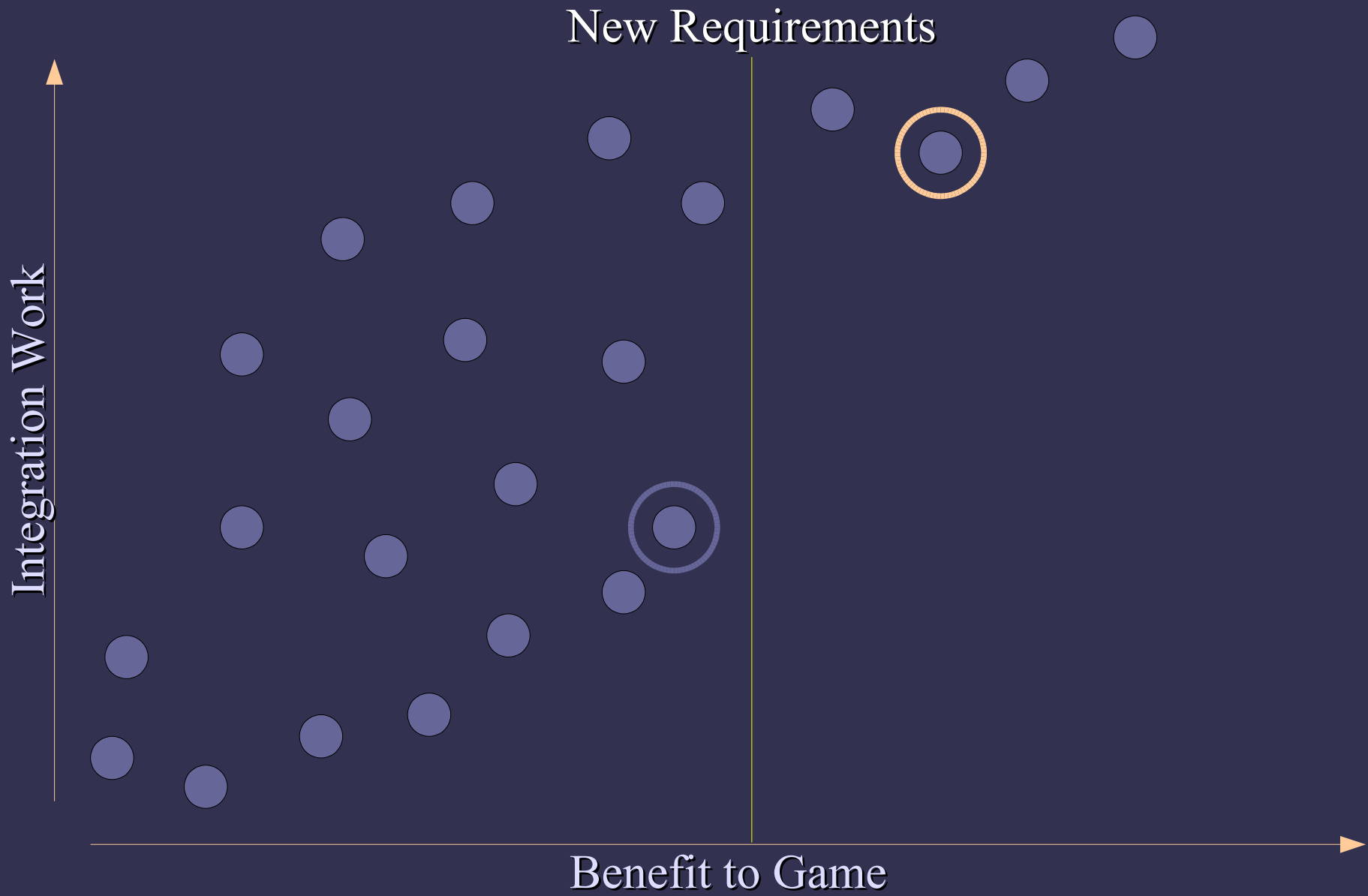


Initial integration

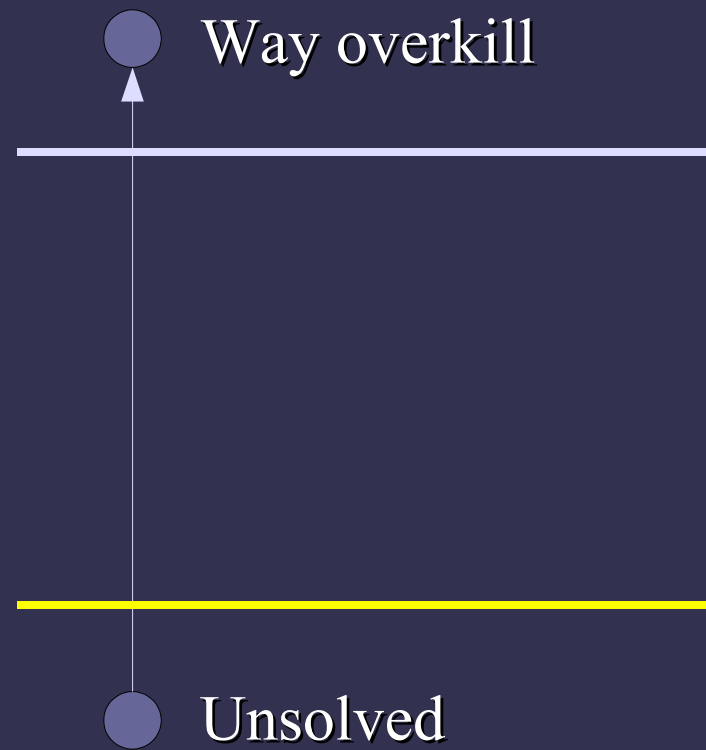
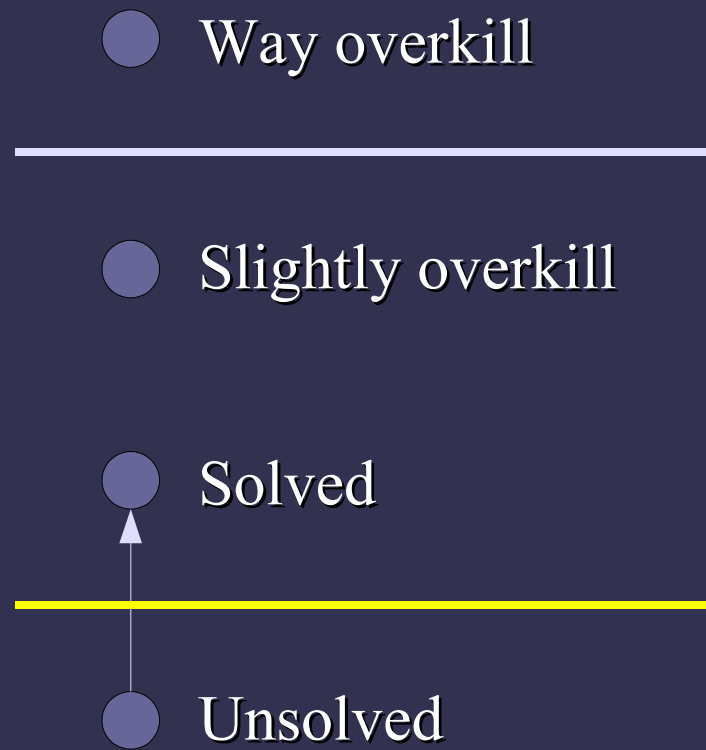
New Requirements



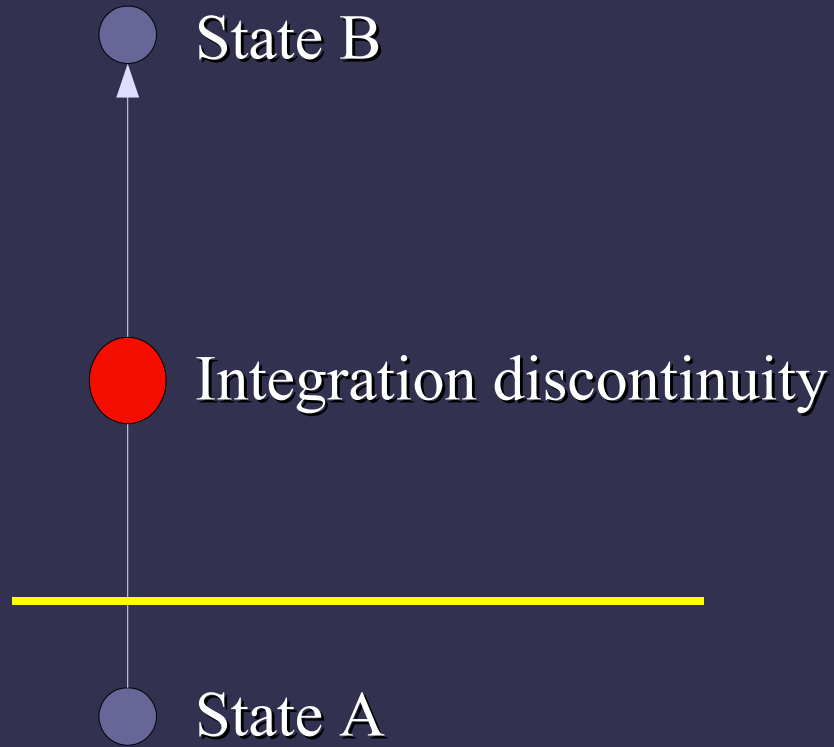
Integration progression



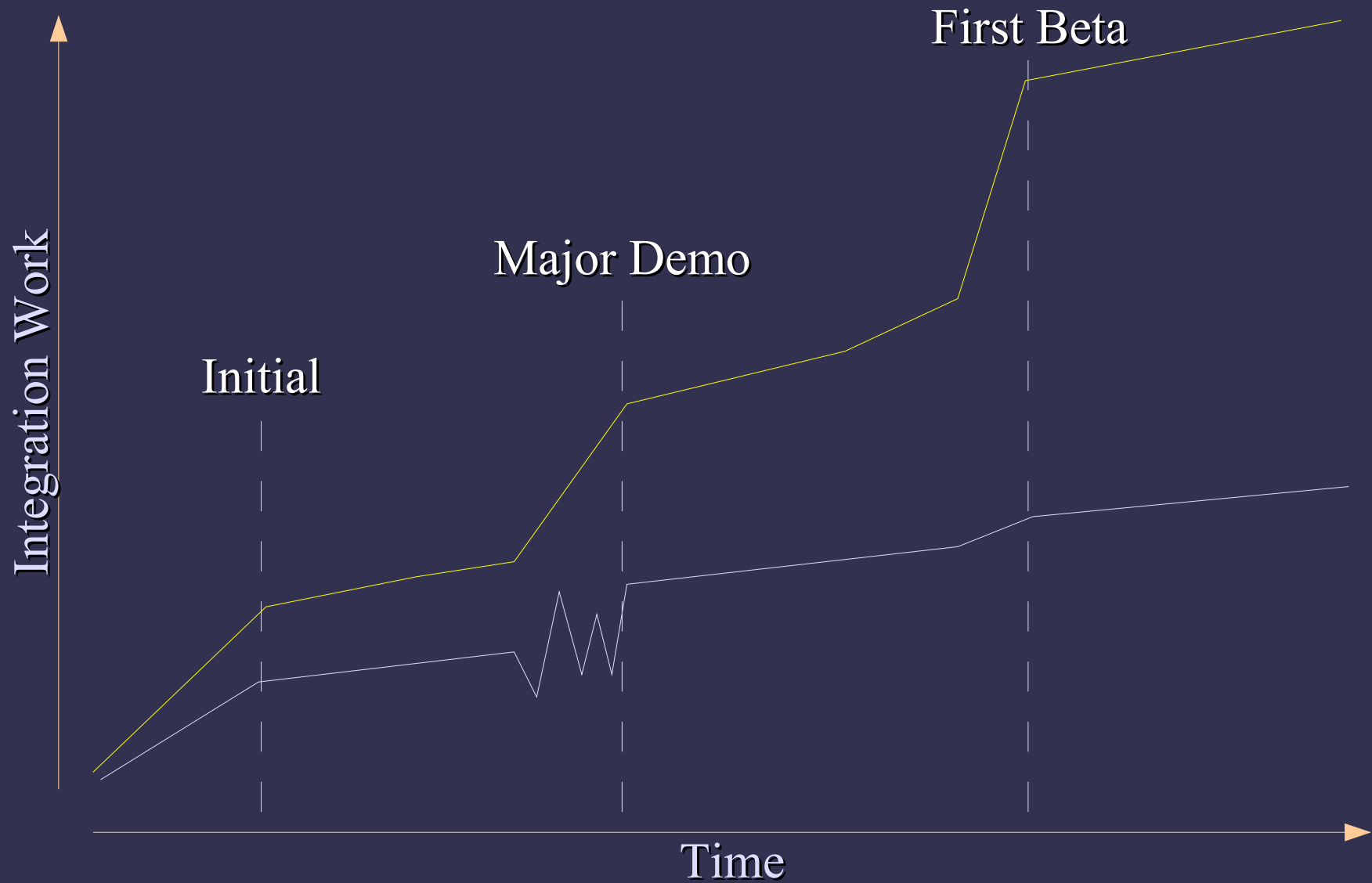
Integration discontinuity



APIs can force large steps



Integration discontinuity



Discontinuities waste work

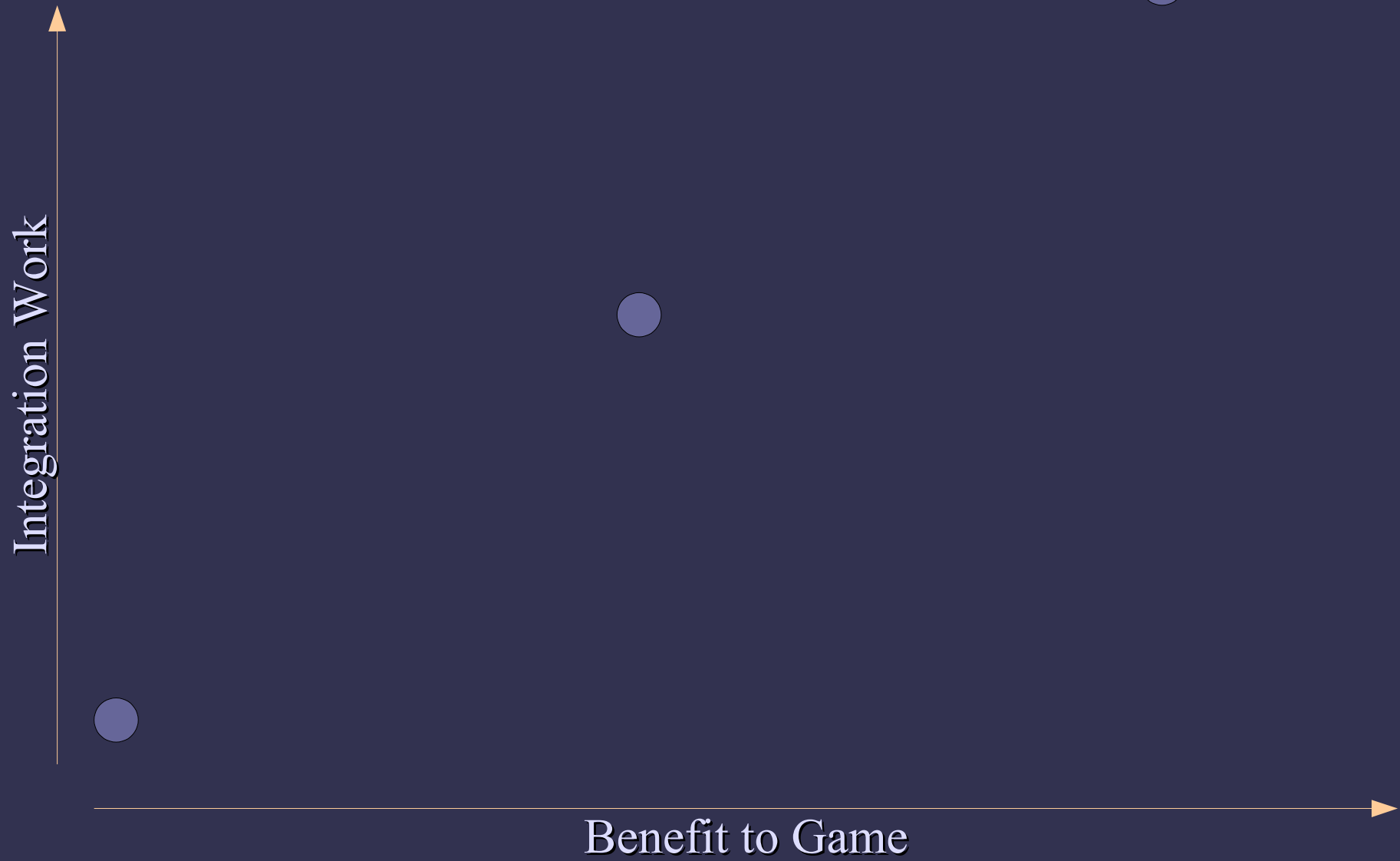
The primary goal of component API design  
is to eliminate API discontinuities.



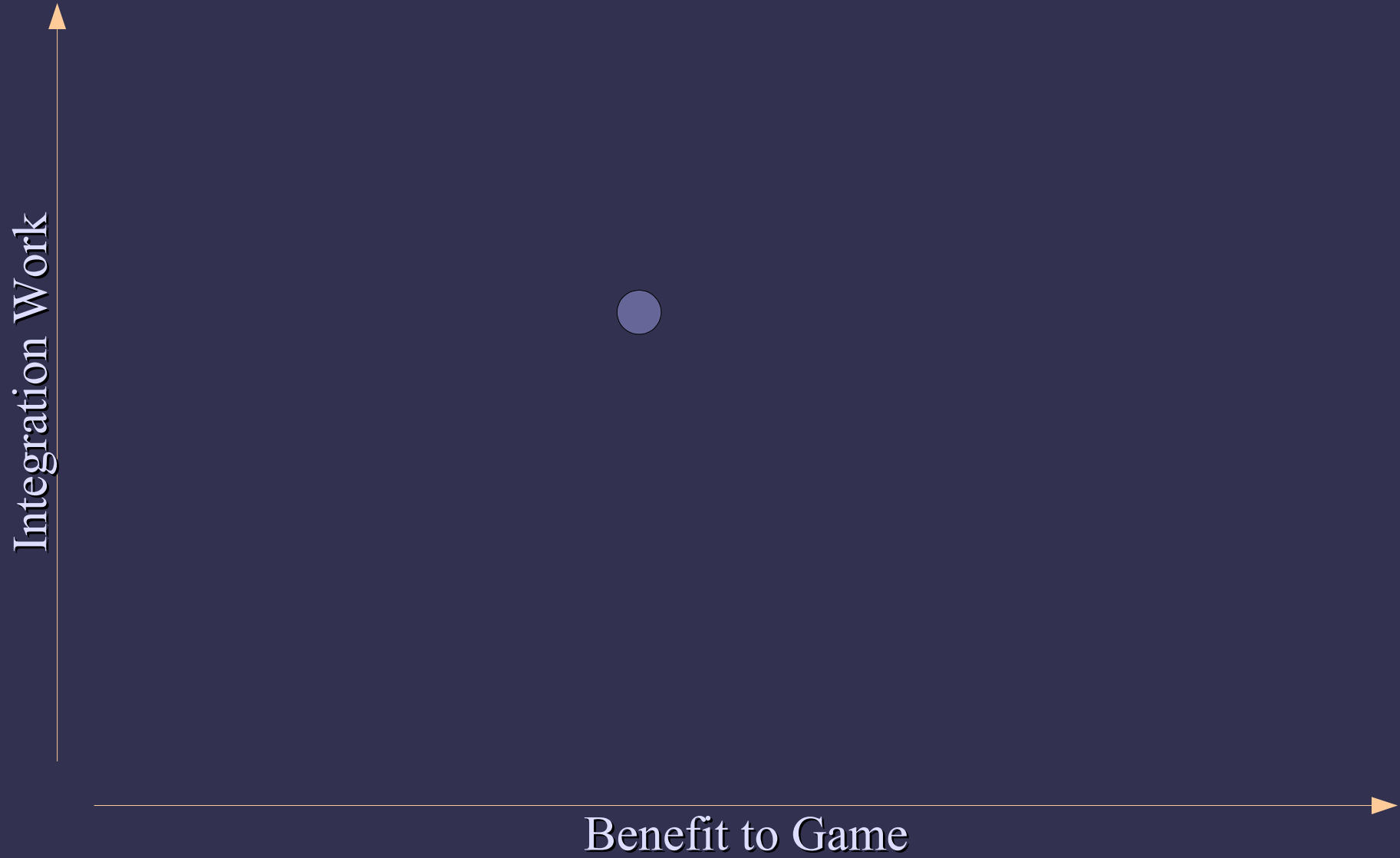
# Current API design trends



Current API design trends



# Current API design trends



# Current API design trends

Granularity - A or BC

Redundancy - A or B

Coupling - A implies B

Retention - A mirrors B

Flow Control - A invokes B

## Five Characteristics

- A `UpdateOrientation(Object);`
- B `Orientation = GetOrientation(Object);`  
`Change = GetOrientationChange(Object);`  
`SetOrientation(Object, Orientation + Change);`
- C `Orientation = GetOrientation(Object);`  
`Change = GetOrientationChange(Object);`  
`Change += 3.14f; // TODO: Close enough to Pi?`  
`SetOrientation(Object, Orientation + Change);`
- D `Orientation = GetOrientation(Object);`  
`Change = GetOrientationChange(Object);`  
`RunSomeOtherUnrelatedThing();`  
`SetOrientation(Object, Orientation + Change);`

Granularity – A or BC

- A SetOrientation3x3(Object, Matrix);
- B SetOrientationQ(Object, Quaternion);
- C IdentityOrientation(Object);  
FaceForwards(Object);
- D OrientInDirection(Object, Vector);  
OrientTowards(Object, Point);
- E NewOrient = GetOrientAndChange(Object);  
SetOrientation(Object, NewOrient);
- F Orient = GetOrientation(Object);  
SetOrientDelta(Object, Orient, Change);

Redundancy – A or B

- A UpdateEverything(Universe);
- B SetTime(GlobalTime);  
UpdateObject(Object);
- C BeginObjectSpecification();  
Object = EndObjectSpecification();
- D String1 = GetMungedName(Name1);  
String2 = GetMungedName(Name2);
- E Object = AllocateAndInitialize();
- F Matrix = MakeMatrixFrom(FloatPointer);  
SetOrientationM(Object, Matrix);
- G Object = ReadObject(Filename);

Coupling – A implies B

- A `SetTime(GlobalTime);`  
`SetPi(3.14f); // TODO: Close enough to Pi?`
  
- B `SetParent(ChildObject, ParentObject);`  
`UpdateOrientation(ChildObject);`
  
- D `SetFileCallbacks(Open, Read, Close);`  
`File = OpenFile(filename);`

Retention – A mirrors B

- A → LibTestNoodleWidgetHit  
LibProcessNoodleWidget  
GameProcessWidgets  
GameUpdate
- B → GameHandleNoodleWidgetHit  
LibTestNoodleWidgetHit  
LibProcessNoodleWidget  
GameProcessWidgets  
GameUpdate
- C → LibNoodleWidgetChangeHeight  
GameHandleNoodleWidgetHit  
LibTestNoodleWidgetHit  
LibProcessNoodleWidget  
GameProcessWidgets  
GameUpdate

Flow Control – A invokes B

A `File = OpenFile(filename);`

B `SetFileCallbacks(Open, Read, Close);`  
`File = OpenFile(filename);`

C 

```
class my_handle : public file_handle
{
public:
    virtual void Open(char *filename);
};
```

D `throw 3.14f; // TODO: stop using exceptions`

## Flow Control – A invokes B

Granularity - A or BC

*Flexibility vs. simplicity*

Redundancy - A or B

*Convenience vs. orthgonality*

Coupling - A implies B

*Less is always better*

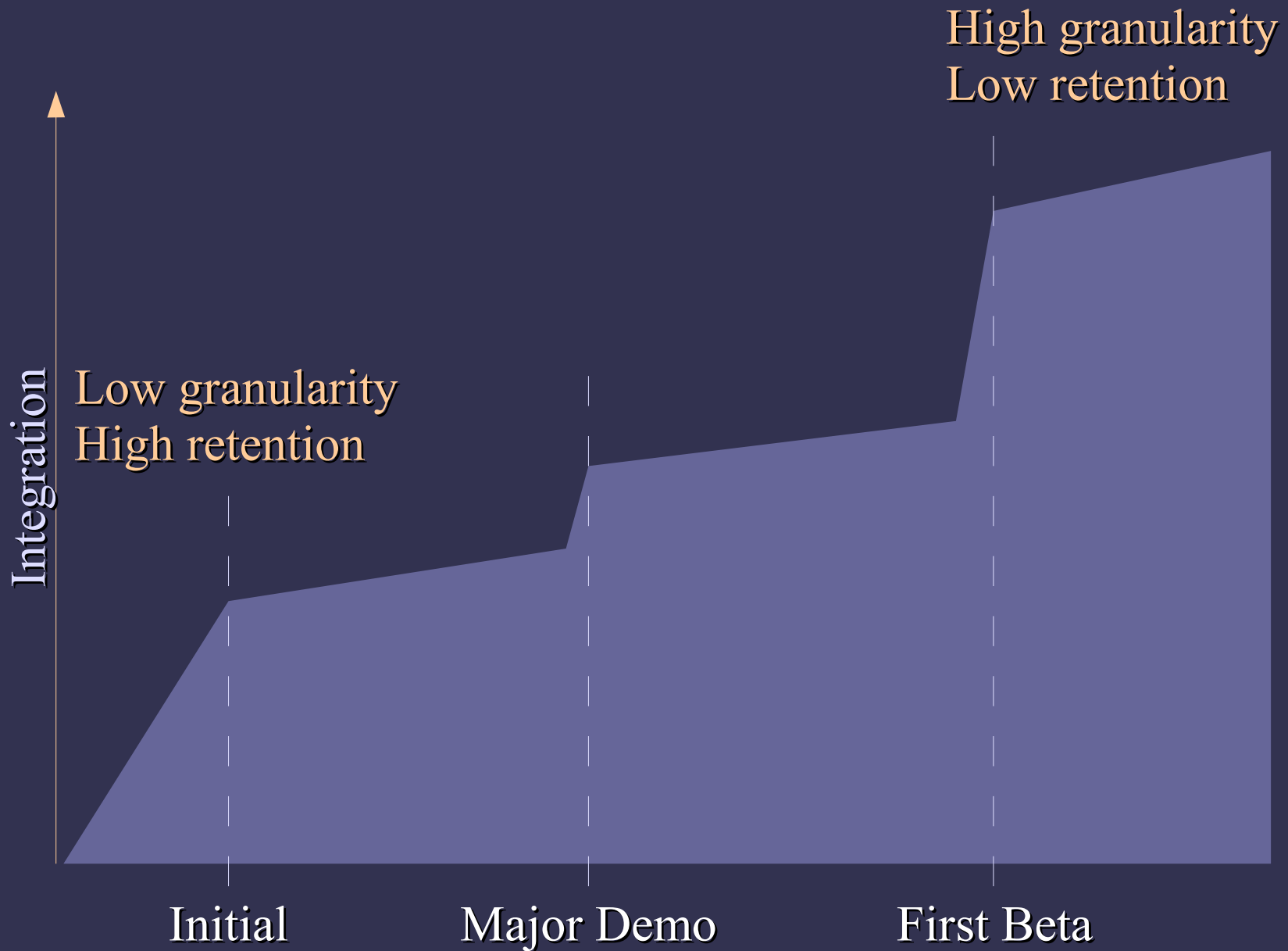
Retention - A equals B

*Synchronization vs. automation*

Flow Control - A invokes B

*More game control is always better*

Recap



Tradeoff decisions often vary

The following examples are based on real-life stories of real game developers in dangerous development situations.

Function names have been changed to protect the  
innocent / guilty.

A `Thing = ReadFile(Filename);`

B `SetFileCallbacks(Open, Read, Close);`  
`Thing = ReadFile(Filename);`

C `Thing = MakeThingFromData(FileData);`

# Game-provided services

```
C  Thing = MakeThingFromData(FileData);

D  FileData = DecompressFile(RawFileData);
   Thing = MakeThingFromData(FileData);
   FreeFileData(FileData);

E  SetMemoryCallbacks(Allocate, Deallocate);
   FileData = DecompressFile(RawFileData);
   Thing = MakeThingFromData(FileData);

F  Size = GetProcessedSize(RawFileData);
   FileData = malloc(Size);
   DecompressFile(RawFileData);
   Thing = MakeThingFromData(FileData);
```

# Game-provided services

```
F  Size = GetProcessedSize(RawFileData);  
    FileData = malloc(Size);  
    ProcessFile(RawFileData, FileData);  
    Thing = MakeThingFromData(FileData);
```

```
G  Thing = NewThing();  
    Read(File, sizeof(Thing), &Thing);
```

```
H  Thing = AllocateInAGP(sizeof(Thing));  
    Read(File, sizeof(Thing), &Thing);
```

## Game-provided services

- A `Inverse = InverseTransform(Xform);`
- B `Xform = TransformFrom(Position, Rotation);`  
`Inverse = InverseTransform(Xform);`  
`CopyTransform(Inverse, Position, Rotation);`
- C `GetPosition(GameXform, Position);`  
`GetRotation(GameXform, Rotation);`  
`Xform = TransformFrom(Position, Rotation);`  
`Inverse = InverseTransform(Xform);`  
`CopyTransform(Xform, Position, Rotation);`  
`SetPosition(GameXform, Position);`  
`SetRotation(GameXform, Rotation);`
- D `InverseTransformQ(Position, Rotation,`  
`Position, Rotation);`

Parameter redundancy

A `UpdateNode(Node);`  
`RenderNode(Node);`

B `// Update`  
`Rotate(XForm, t * RadiansPerSecond);`  
`Translate(XForm, t * MetersPerSecond);`  
`WorldMesh = BuildWorldState(Mesh, ParentMesh);`  
  
`// Render`  
`MaterialSort = NewMaterialSort();`  
`Sort(MaterialSort, WorldMesh);`  
`RenderOpaque(WorldMesh, XForm, MaterialSort);`  
`RenderAlpha(WorldMesh, XForm, MaterialSort);`  
`ReleaseMaterialSort(MaterialSort);`

# Granularity transitions

```
C // Update
  Rotate(XForm, t * RadiansPerSecond);
  Translate(XForm, t * MetersPerSecond);
  WorldMesh = BuildWorldState(Mesh, ParentMesh);

  // Render
  Render(WorldMesh, XForm);

D // Update
  Rotate(XForm, t * RadiansPerSecond);
  Translate(XForm, t * MetersPerSecond);
  WorldMesh = BuildWorldState(Mesh, ParentMesh);

  // Render
  MaterialSort = NewMaterialSort();
  RenderSorted(WorldMesh, XForm, MaterialSort);
  ReleaseMaterialSort(MaterialSort);
```

## Granularity transitions

```
A Rocket = CreateRigidBody();  
Pole = CreateRigidBody();  
Hookline = CreateJoint(Rocket, Pole);  
Simulate();
```

```
B if(XButtonDown)  
{  
    if(!Hookline)  
    {  
        Hookline = CreateJoint(Rocket, Pole);  
    }  
}  
else if(Hookline)  
{  
    DeleteJoint(Hookline);  
    Hookline = 0;  
}  
Simulate();
```

# Retention mismatch

```
B  if(XButtonDown)
    {
        if(!Hookline)
        {
            Hookline = CreateJoint(Rocket, Pole);
        }
    }
else if(Hookline)
{
    DeleteJoint(Hookline);
    Hookline = 0;
}
Simulate();
```

```
C  if(XButtonDown) DoJoint(Rocket, Pole);
Simulate();
```

# Retention mismatch

Optimizing the five characteristics leads to an API  
which is gradually tiered, highly decoupled,  
has no retention at its most granular tier,  
and always lets the game dictate the flow of control.

And now, the crib sheet.

- ✓ The first thing you did was write the code that uses the API.
- ✓ The second thing you did was write the code that uses the API.

API evaluation checklist

- ✓ All retained mode constructs have immediate-mode equivalents
- ✓ For every API that uses callbacks or inheritance, there is an equivalent API that does neither.
- ✓ No API requires the use of a API-specific datatype for which the average game already has an equivalent.
- ✓ Any API function your game may not consider atomic can be re-written using between 2 and 4 more granular APIs (not counting accessors)

## API evaluation checklist

- ✓ Any data which does not clearly have a reason for being opaque should be transparent in all possible ways (construction, access, I/O, etc.)
- ✓ Use of the component's resource management (memory, file, string, etc.) is completely optional.
- ✓ Use of the component's file format is completely optional.
- ✓ Full run-time source code is available.

## API evaluation checklist

# Designing and Evaluating Reusable Components

Casey Muratori

[casey@mollyrocket.com](mailto:casey@mollyrocket.com)

Questions and answers

