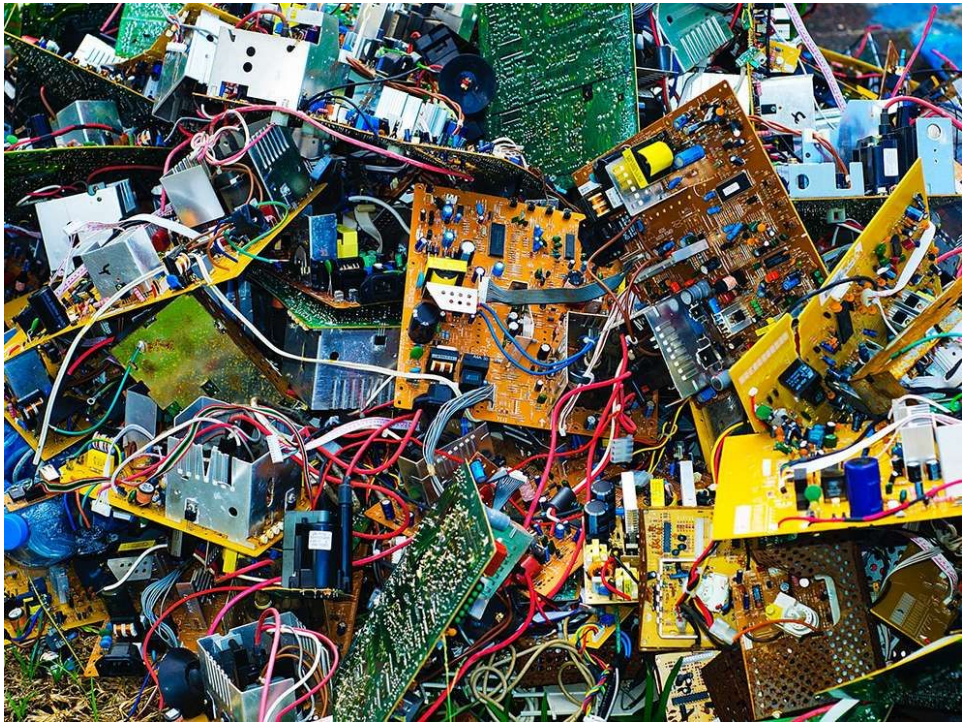
Game Architecture

Data Oriented Design

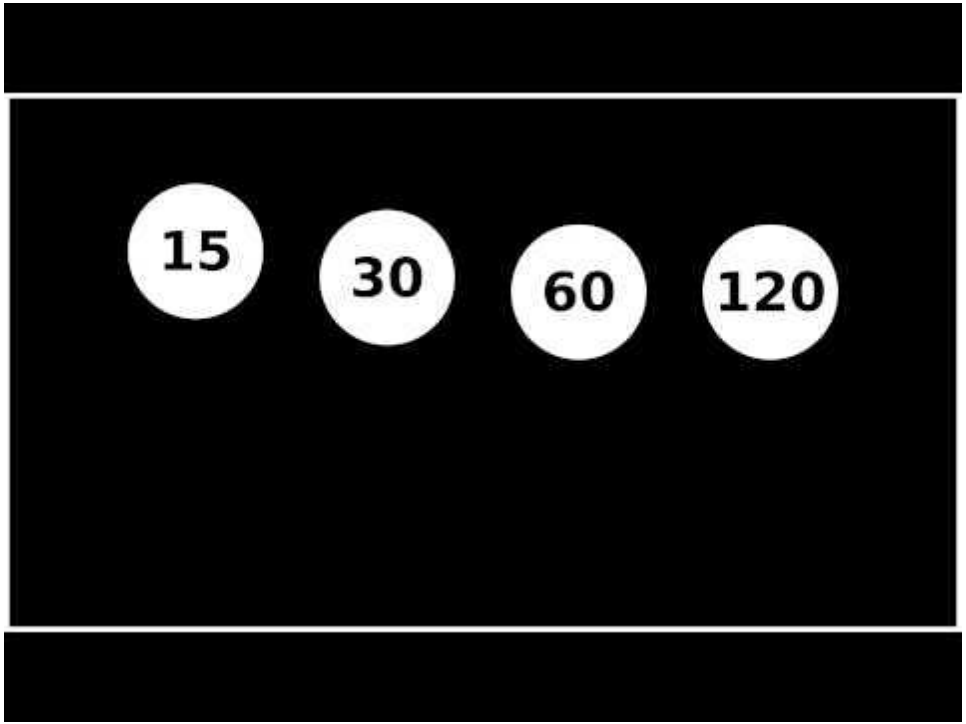
Today’s Agenda

- Motivation
- Review Hardware
- Data oriented design

Hardware is reality



Framerate!



But...

“Premature optimization is the root of all evil.”

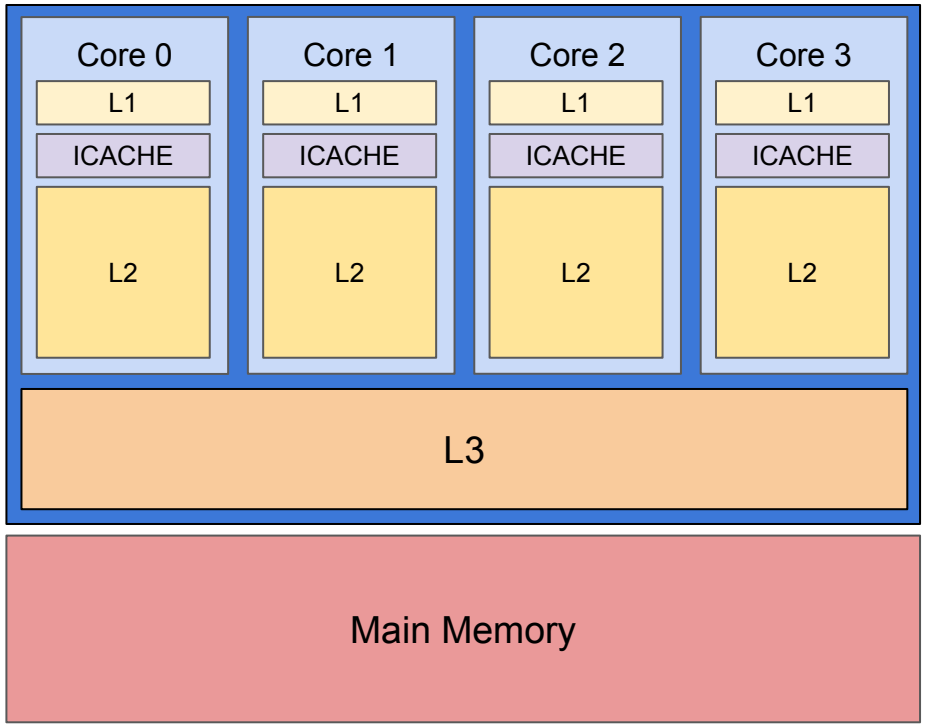
But...

“We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.”

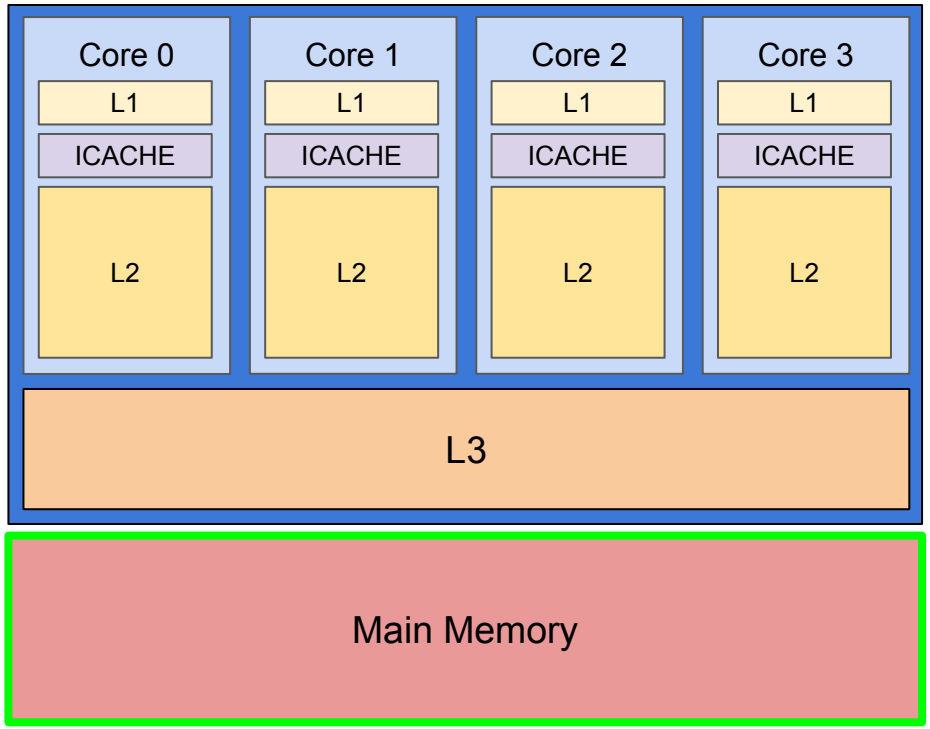
But...

```
// From: https://github.com/id-Software/Quake-III-Arena
float Q_rsqrt( float number )
{
	long i;
	float x2, y;
	const float threehalfs = 1.5F;

	x2 = number * 0.5F;
	y  = number;
	i  = * ( long * ) &y;                       // evil floating point bit level hacking
	i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
	y  = * ( float * ) &i;
	y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//	y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed

	return y;
}
```

Comp Org for Game Developers

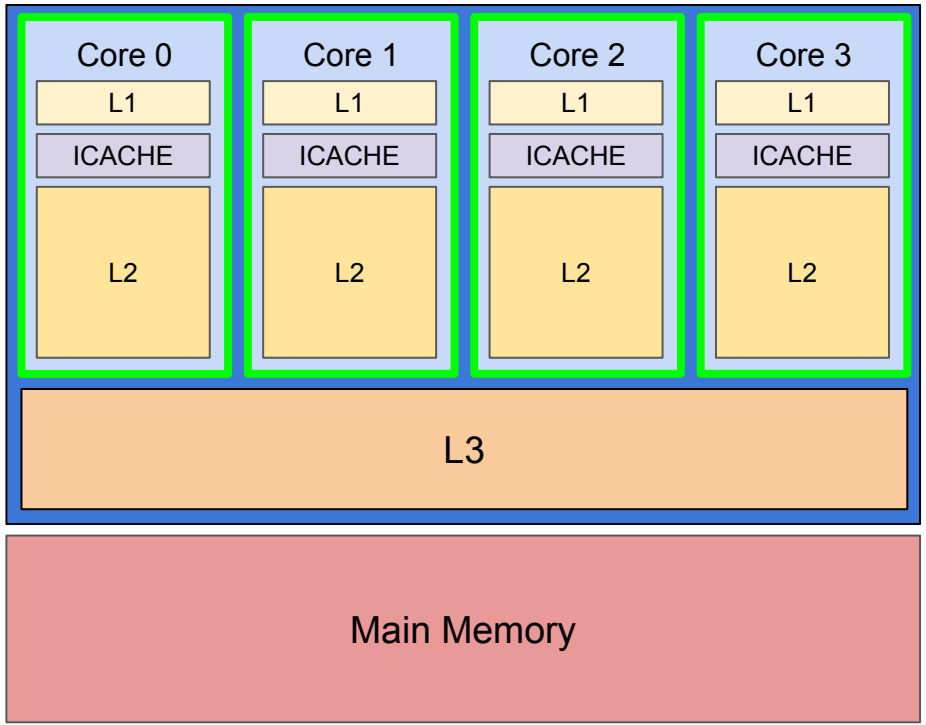


Comp Org for Game Developers



Size	8 gb
Latency	300 cycles

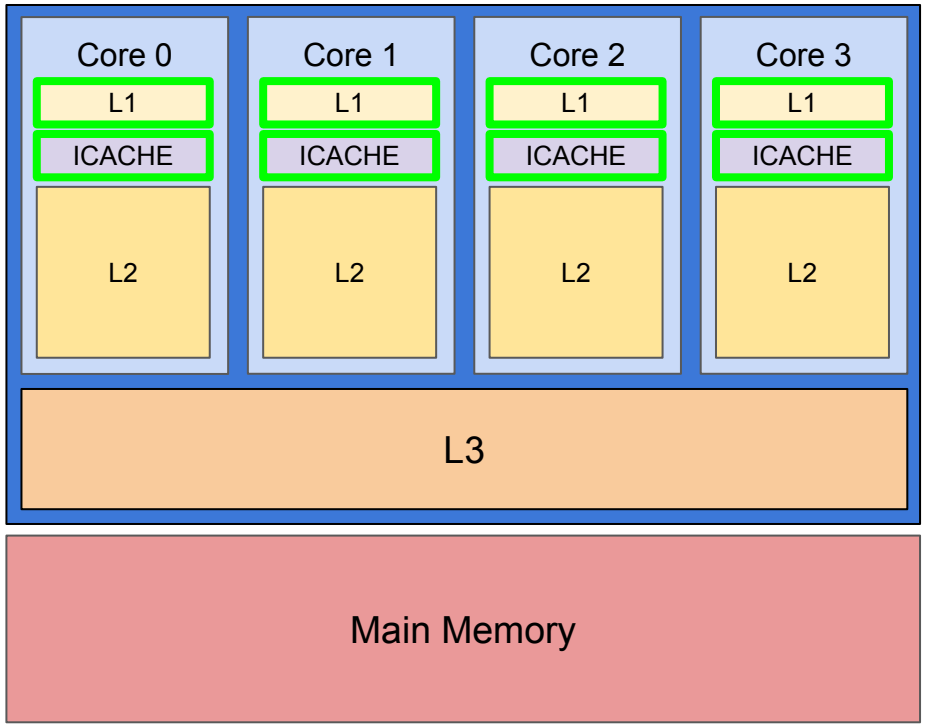
Core



Cores: 4

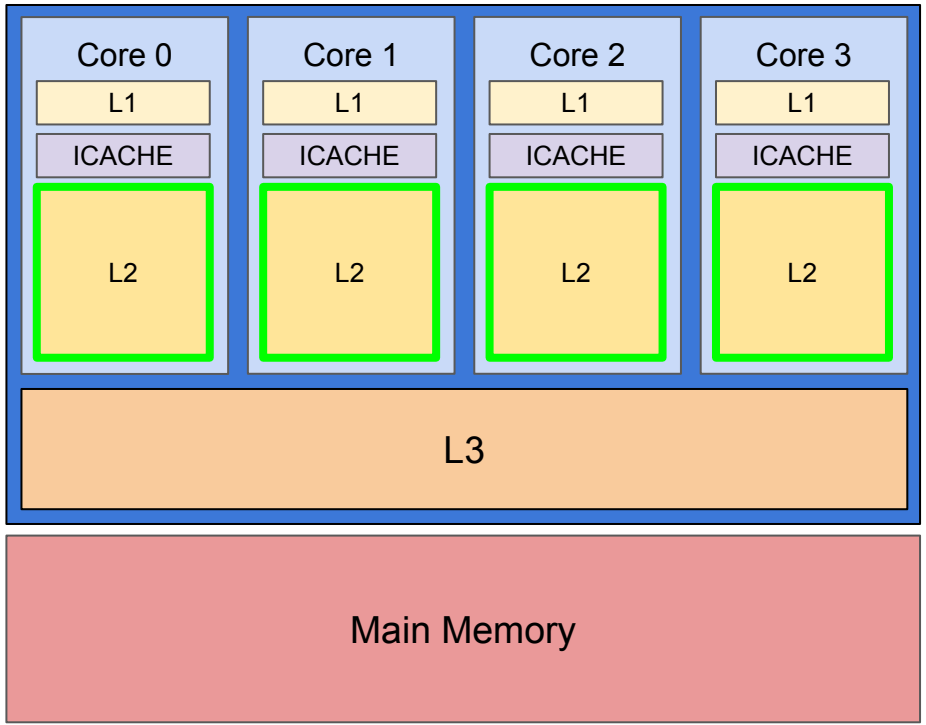
Clock Frequency: 3.6 ghz

Comp Org for Game Developers



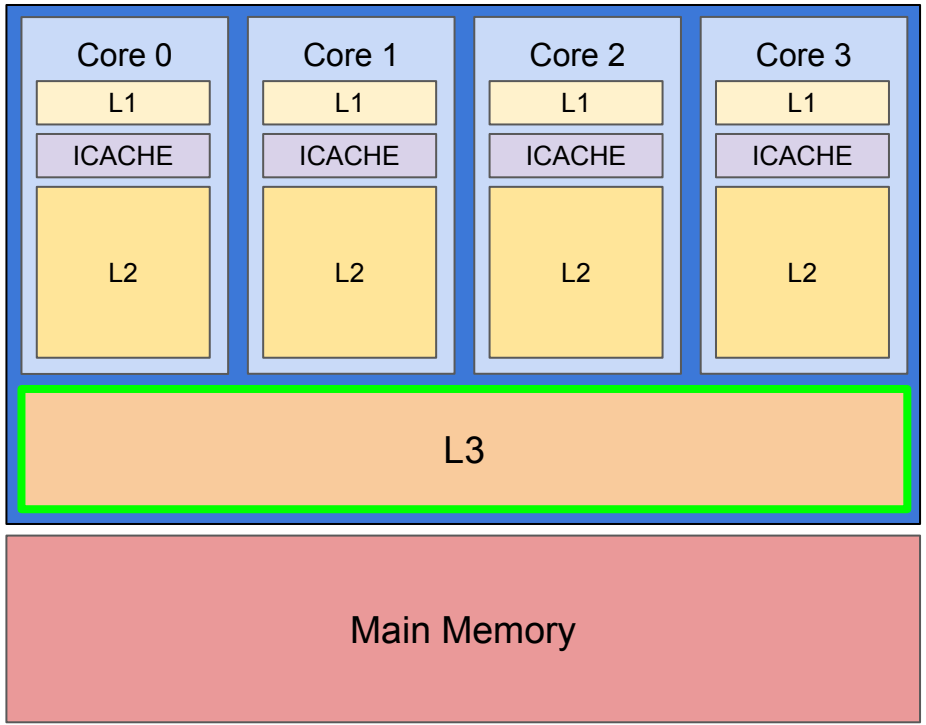
Size: 32 kb
Latency: 4 cycles

Comp Org for Game Developers



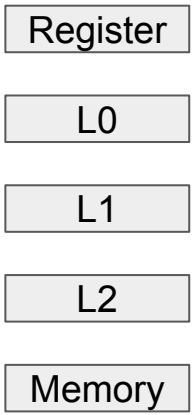
Size: 256 kb
Latency: 12 cycles

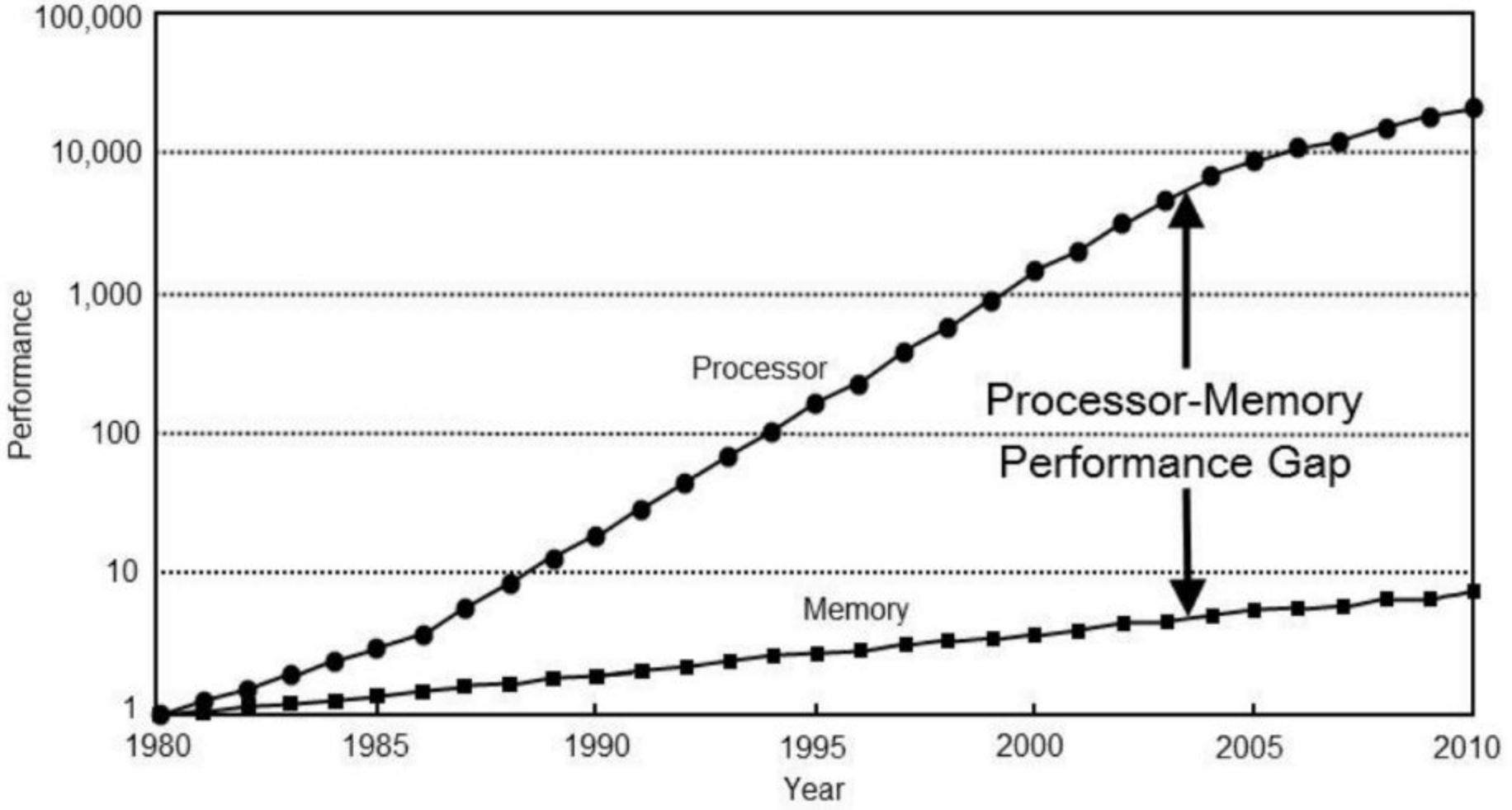
Comp Org for Game Developers



Size: 8 mb
Latency: 30 cycles

Latency



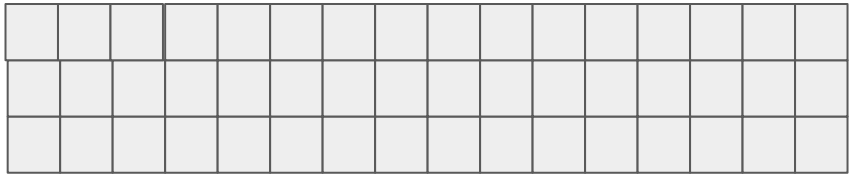

100,000
10,000
1,000
100
10
1
Performance
Processor
Memory
Processor-Memory
Performance Gap
1980
1985
1990
1995
2000
2005
2010
Year

Example

```
class bot {
  vec3 _position;
  ma4 _transform;
  float _aim_offset;
  mat4 _previous_transform;
  float _aim_angle;

  void update_aim(vec3 target) {
    _aim_angle = dot(_position, target) * _aim_offset;
  }
};
```

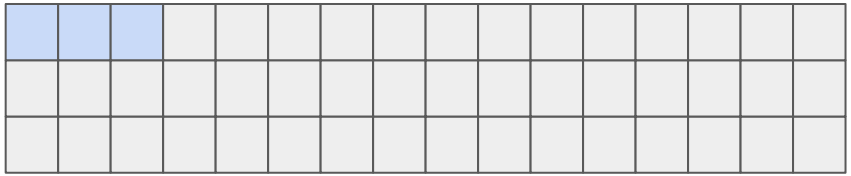
Example

```
class bot {
  vec3 _position;
  ma4 _transform;
  float _aim_offset;
  mat4 _previous_transform;
  float _aim_angle;

  void update_aim(vec3 target) {
    _aim_angle = dot(_position, target) * _aim_offset;
  }
};
```



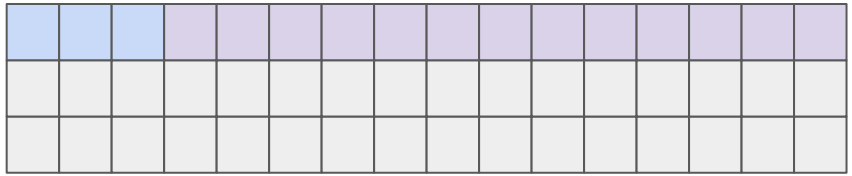
Example

```
class bot {
  vec3 _position;
  ma4 _transform;
  float _aim_offset;
  mat4 _previous_transform;
  float _aim_angle;

  void update_aim(vec3 target) {
    _aim_angle = dot(_position, target) * _aim_offset;
  }
};
```



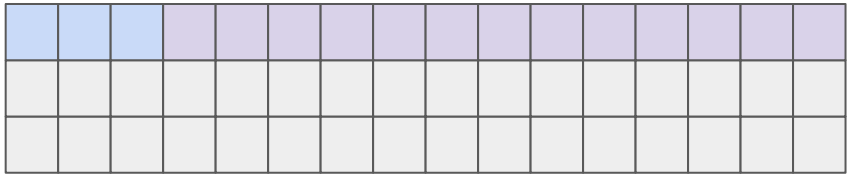
Example

```
class bot {
  vec3 _position;
  ma4 _transform;
  float _aim_offset;
  mat4 _previous_transform;
  float _aim_angle;

  void update_aim(vec3 target) {
    _aim_angle = dot(_position, target) * _aim_offset;
  }
};
```



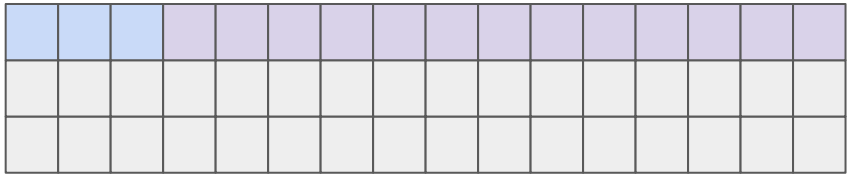
Example

```
class bot {
  vec3 _position;
  ma4 _transform;
  float _aim_offset;
  mat4 _previous_transform;
  float _aim_angle;

  void update_aim(vec3 target) {
    _aim_angle = dot(_position, target) * _aim_offset;
  }
};
```



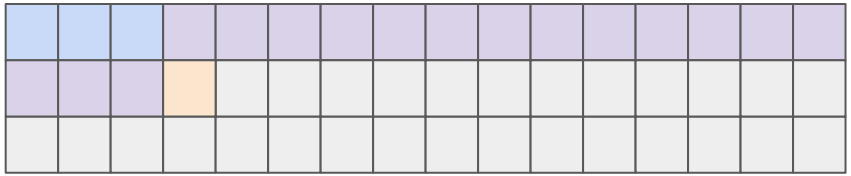
Example

```
class bot {
  vec3 _position;
  ma4 _transform;
  float _aim_offset;
  mat4 _previous_transform;
  float _aim_angle;

  void update_aim(vec3 target) {
    _aim_angle = dot(_position, target) * _aim_offset;
  }
};
```



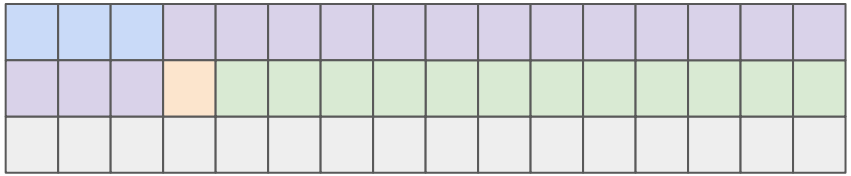
Example

```
class bot {
  vec3 _position;
  ma4 _transform;
  float _aim_offset;
  mat4 _previous_transform;
  float _aim_angle;

  void update_aim(vec3 target) {
    _aim_angle = dot(_position, target) * _aim_offset;
  }
};
```



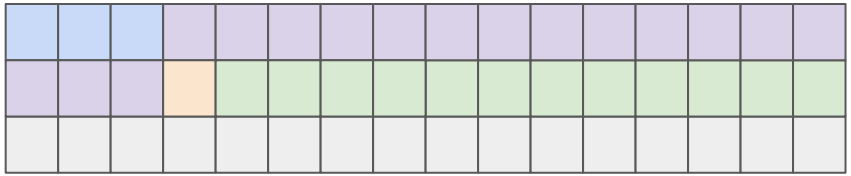
Example

```
class bot {
  vec3 _position;
  ma4 _transform;
  float _aim_offset;
  mat4 _previous_transform;
  float _aim_angle;

  void update_aim(vec3 target) {
    _aim_angle = dot(_position, target) * _aim_offset;
  }
};
```



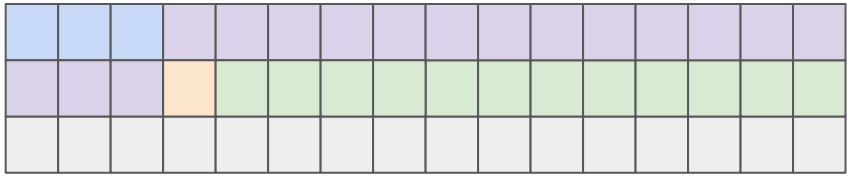
Example

```
class bot {
  vec3 _position;
  ma4 _transform;
  float _aim_offset;
  mat4 _previous_transform;
  float _aim_angle;

  void update_aim(vec3 target) {
    _aim_angle = dot(_position, target) * _aim_offset;
  }
};
```



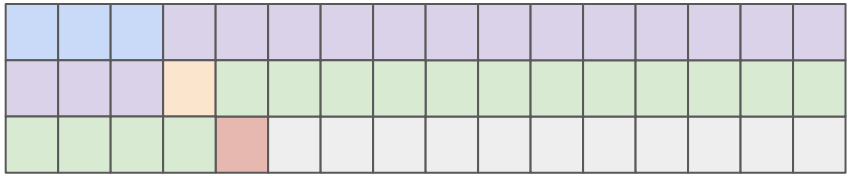
Example

```
class bot {
  vec3 _position;
  ma4 _transform;
  float _aim_offset;
  mat4 _previous_transform;
  float _aim_angle;

  void update_aim(vec3 target) {
    _aim_angle = dot(_position, target) * _aim_offset;
  }
};
```



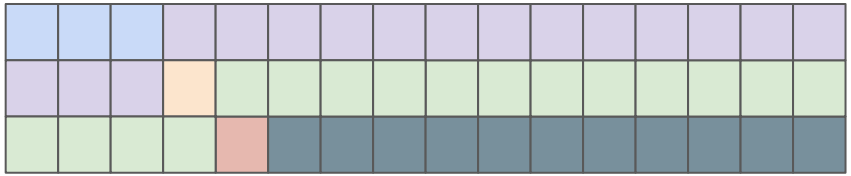
Example

```
class bot {
  vec3 _position;
  ma4 _transform;
  float _aim_offset;
  mat4 _previous_transform;
  float _aim_angle;

  void update_aim(vec3 target) {
    _aim_angle = dot(_position, target) * _aim_offset;
  }
};
```



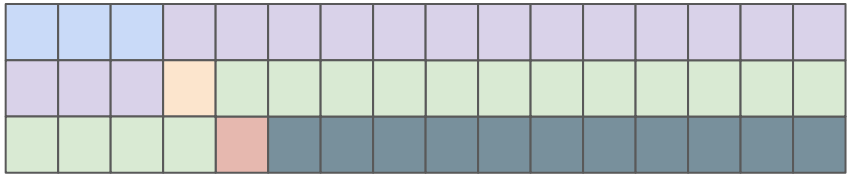
Example

```
class bot {
  vec3 _position;
  ma4 _transform;
  float _aim_offset;
  mat4 _previous_transform;
  float _aim_angle;

  void update_aim(vec3 target) {
    _aim_angle = dot(_position, target) * _aim_offset;
  }
};
```



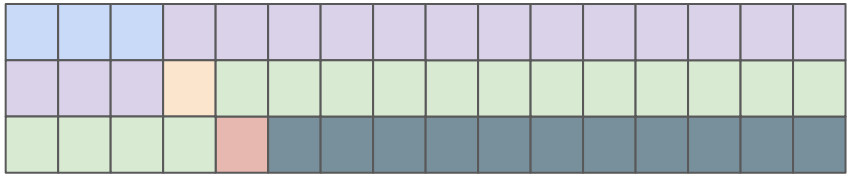
Example

```
class bot {
  vec3 _position;
  ma4 _transform;
  float _aim_offset;
  mat4 _previous_transform;
  float _aim_angle;

  void update_aim(vec3 target) {
    _aim_angle = dot(_position, target) * _aim_offset;
  }
};
```



Example

```
class bot {
  vec3 _position;
  ma4 _transform;
  float _aim_offset;
  mat4 _previous_transform;
  float _aim_angle;

  void update_aim(vec3 target) {
    _aim_angle = dot(_position, target) * _aim_offset;
  }
};
```

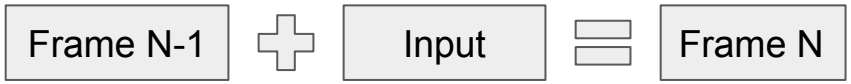


No magic wand

No, the compiler can not do it for you!

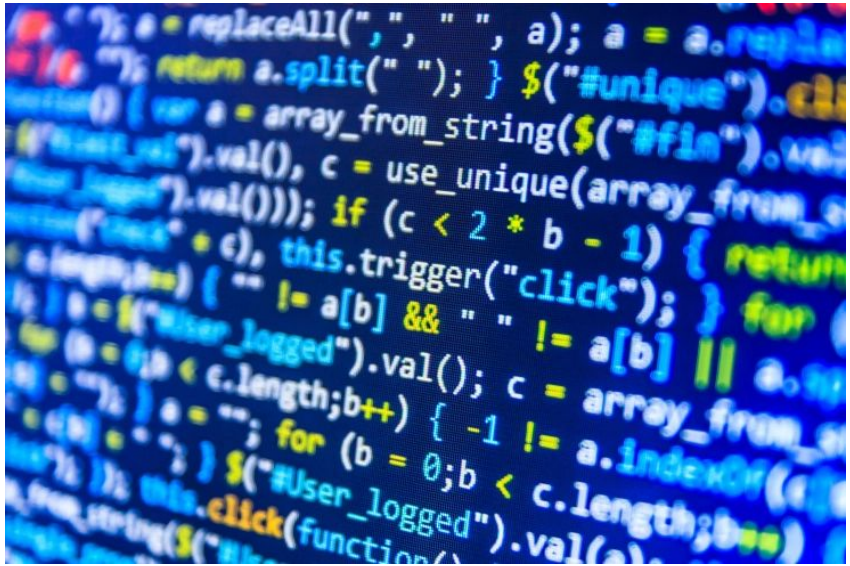
Data Oriented Design

The goal of a program - and every part of a program - is to transform data from one form to another.



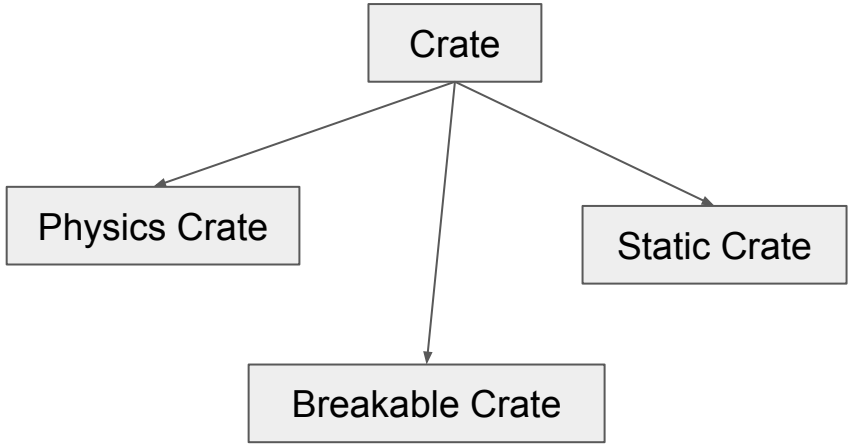
Data Oriented Design

All problems are data problems. If you don’t understand the data than you don’t fully understand the problem.



Principles

Don’t solve problems you do not have.



Principles

Where there is one there are many.

```
void update_physics(entity e) {
  bool paused = is_physics_paused()
  If (!paused) {
    // Do stuff with e
  }
}
```

Principles

Where there is one there are many.

```
void update_physics(vector<entity> es) {
  bool paused = is_physics_paused()
  If (!paused) {
    for (auto& e : es) {
      // Do stuff with e
    }
  }
}
```

Principles

Where there is one there are many.

```
class bot {
  vec3 _position;
  ma4 _transform;
  float _aim_offset;
  mat4 _previous_transform;
  float _aim_angle;

  void update_aim(vec3 target) {
    _aim_angle = dot(_position, target) * _aim_offset;
  }
};
```

Principles

Where there is one there are many.

```
class bot {
  vec3 _position;
  float _aim_offset;
  float _aim_angle;
  ma4 _transform;
  mat4 _previous_transform;

  void update_aim(vec3 target) {
    _aim_angle = dot(_position, target) * _aim_offset;
  }
};
```

Principles

Where there is one there are many.

```
struct bot_in {
  vec3 _position;
  float _aim_offset;
};
struct bot_out {
  float _aim_angle;
};
void update_aims(vec3 target, bot_in* in, bot_out* out, int c) {
  for (int i = 0; i < c; ++i) {
    out[i]._aim_angle = dot(target, in[i]._position);
    out[i[._aim_angle += in._aim_offset;
  }
}
```

Principles

Where there is one there are many.

```
struct bot_in {
  vec3 _position;
  float _aim_offset;
};
struct bot_out {
  float _aim_angle;
};
void update_aims(vec3 target, bot_in* in, bot_out* out, int c) {
  for (int i = 0; i < c; ++i) {
    out[i]._aim_angle = dot(target, in[i]._position);
    out[i[._aim_angle += in._aim_offset;
  }
}
```

Principles

The more data you have the more context you have which can help you make better decisions. Don’t throw away context!

```
void transform_point(vec3 point, bool local) {
        If (local) {
                // ...
        }
        else {
                // ...
        }
}

void transform_point_local(vec3 point)
void transform_point_world(vec3 point)
```

Principles

The more data you have the more context you have which can help you make better decisions. Don’t throw away context!

```
struct sword : public weapon {
  virtual void attack() override;
};
struct club: public weapon  {
  virtual void attack() override;
}

weapon_list.push_back(new sword);
weapon_list.push_back(new club);

for (auto weapon : weapon_list) {
  weapon->attack();
}
```

Principles

The more data you have the more context you have which can help you make better decisions. Don’t throw away context!

```
struct sword {
  void attack();
};
struct club {
  void attack();
};

sword_list.push_back(new sword);
club_list.push_back(new club);

for (auto sword : sword _list) {
  sword->attack();
}
for (auto club : club_list) {
  club->attack();
}
```

Conclusion

Design the data. Not the code.

- Better performance
- Easier to debug
- Easier to maintain
- Easier to change