Better Code

Sean Parent | Principal Scientist

Bē
GMUNK

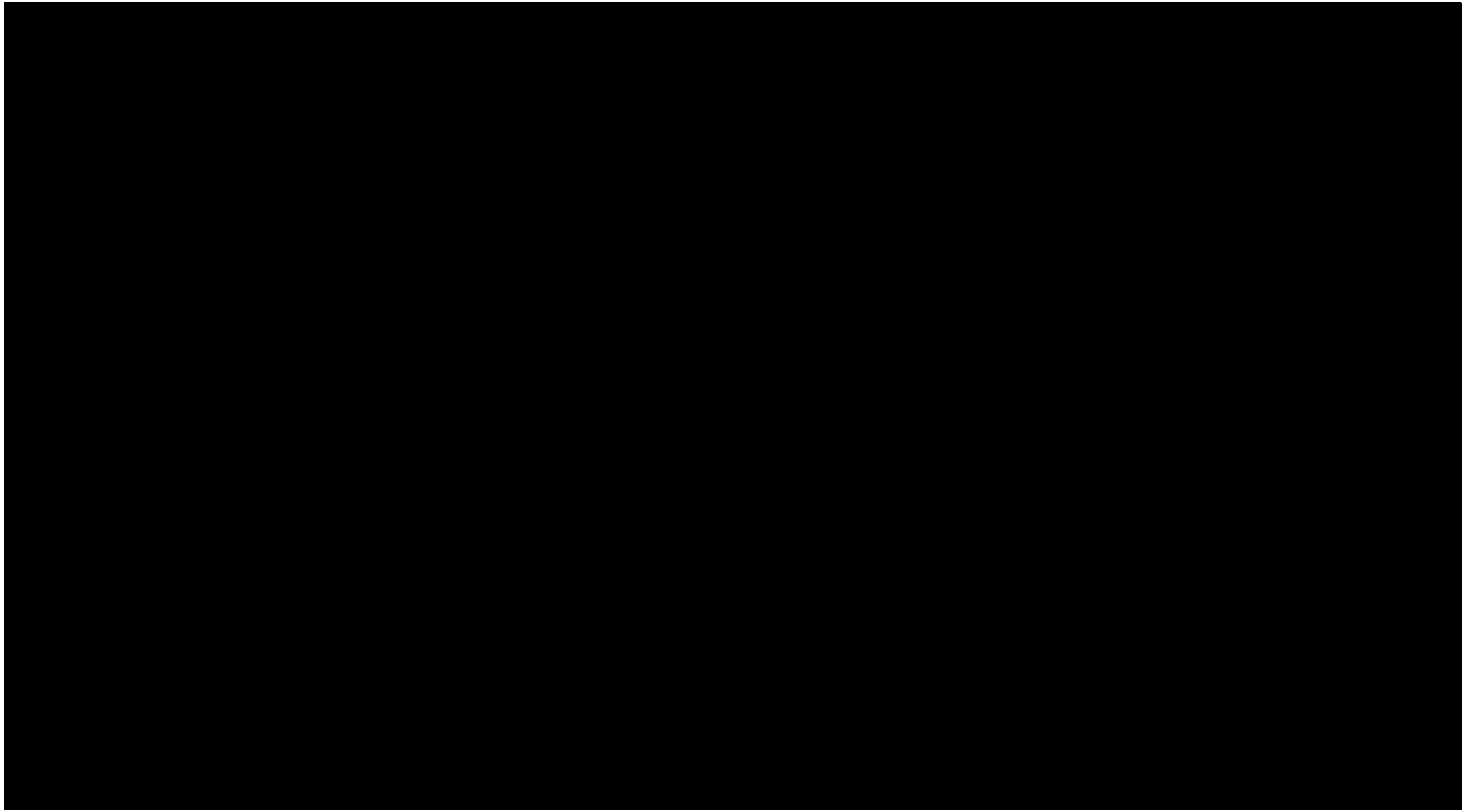
Better Code

- Regular Types
  - Goal: No Incomplete Types
- Algorithms
  - Goal: No Raw Loops
- Data Structures
  - Goal: No Incidental Data Structures
- Runtime Polymorphism
  - Goal: No Raw Pointers
- Concurrency
  - Goal: No Raw Synchronization Primitives

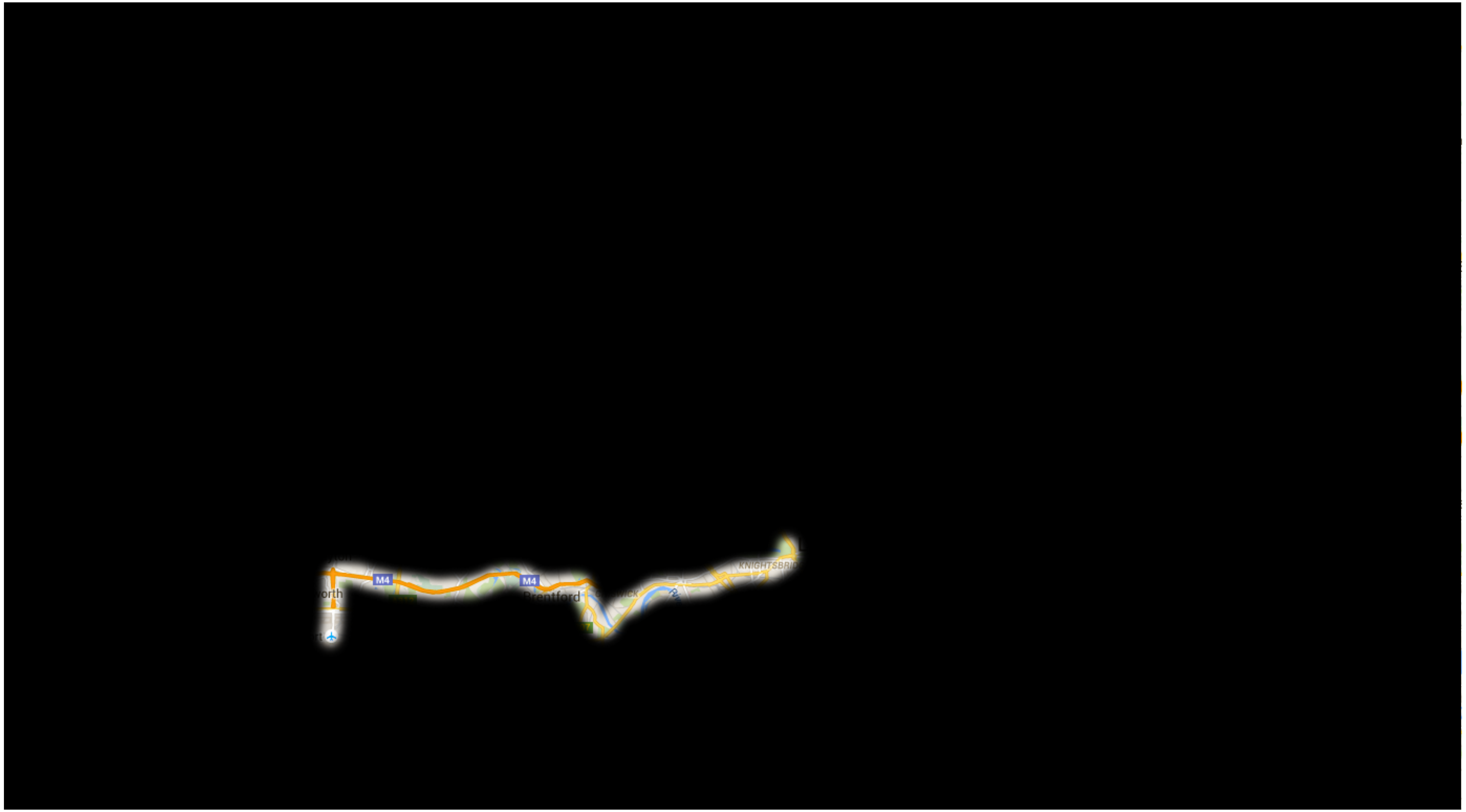
https://github.com/sean-parent/sean-parent.github.io/wiki/Papers-and-Presentations

© 2016 Adobe Systems Incorporated. All Rights Reserved.

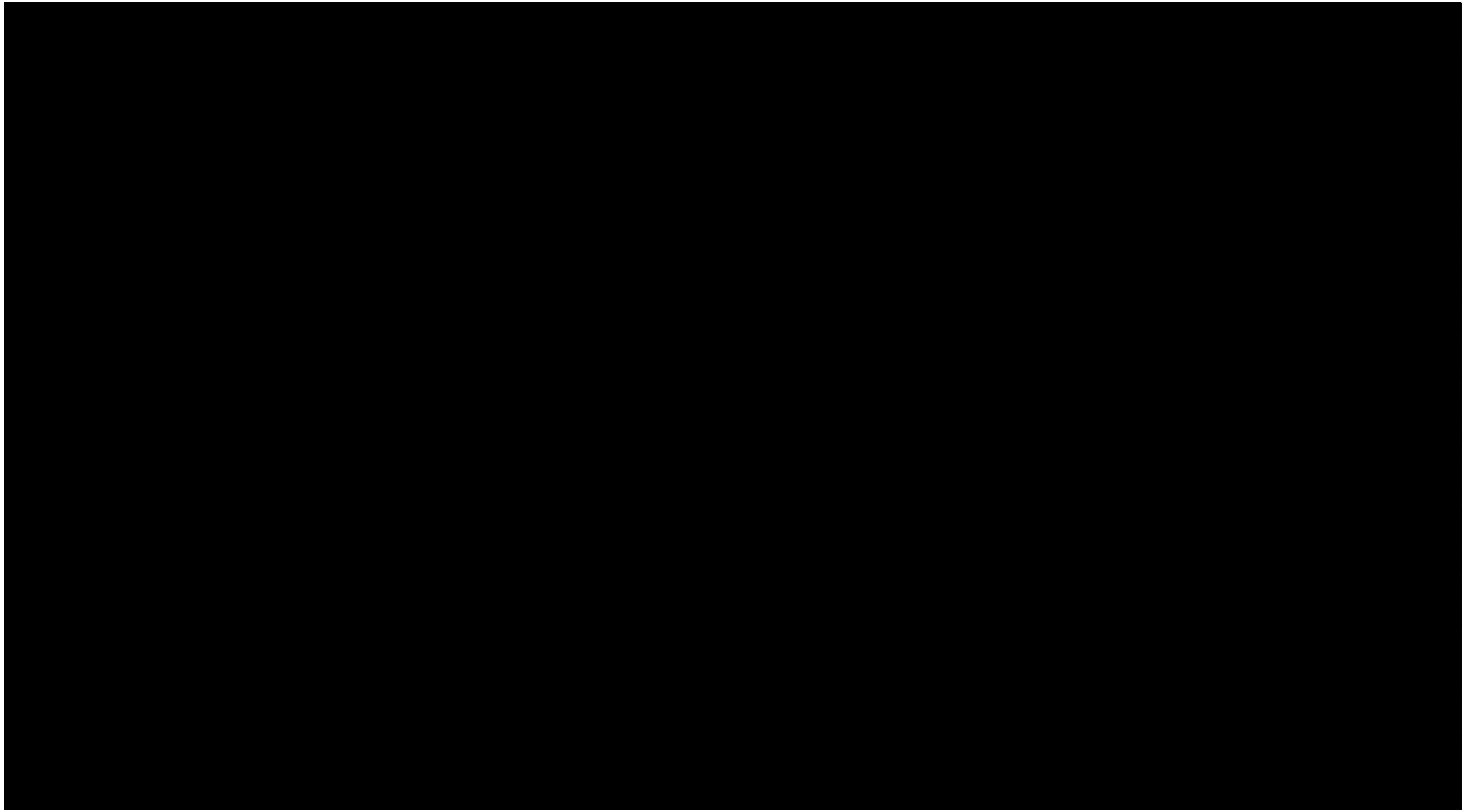
The Knowledge



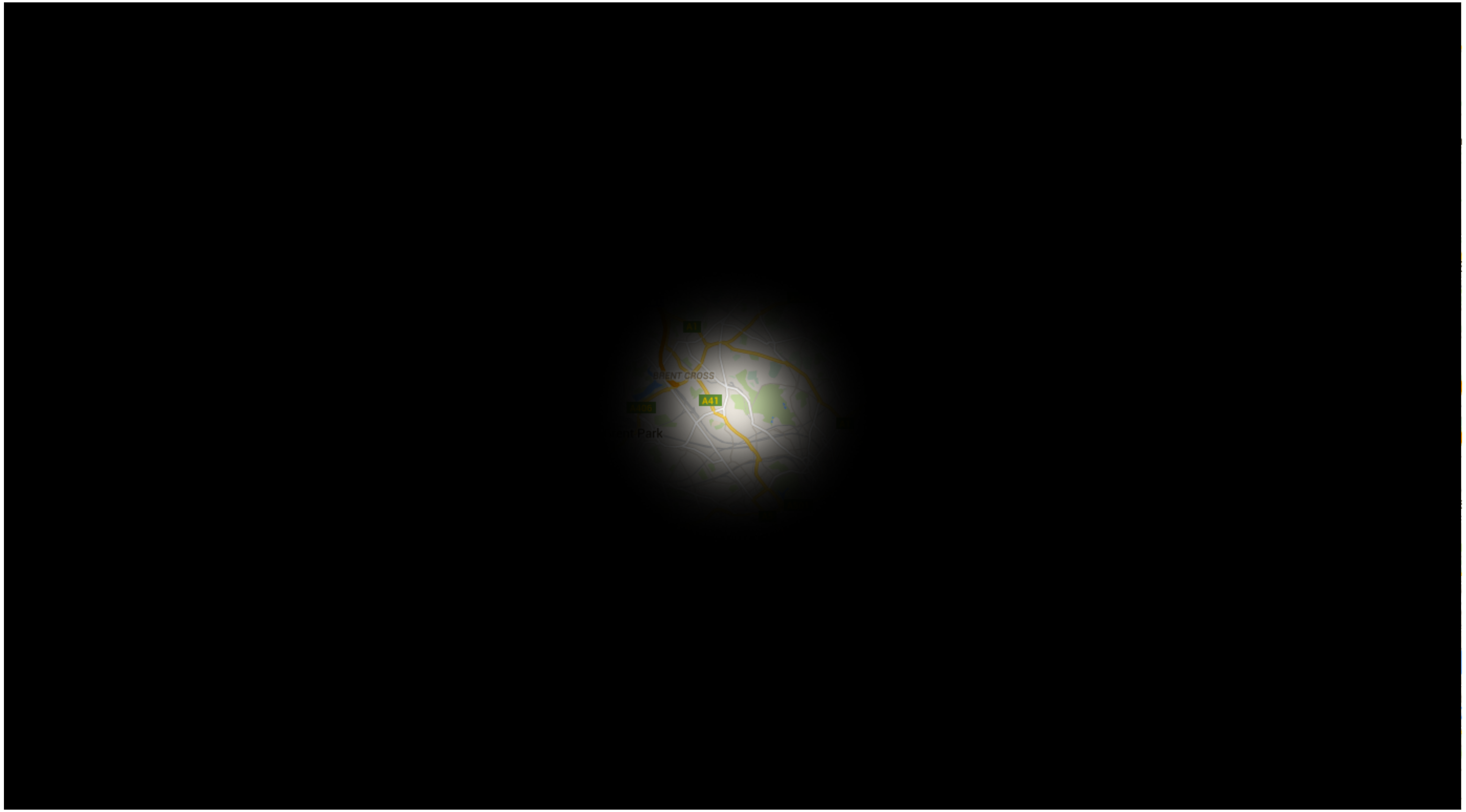
© 2016 Adobe Systems Incorporated. All Rights Reserved.

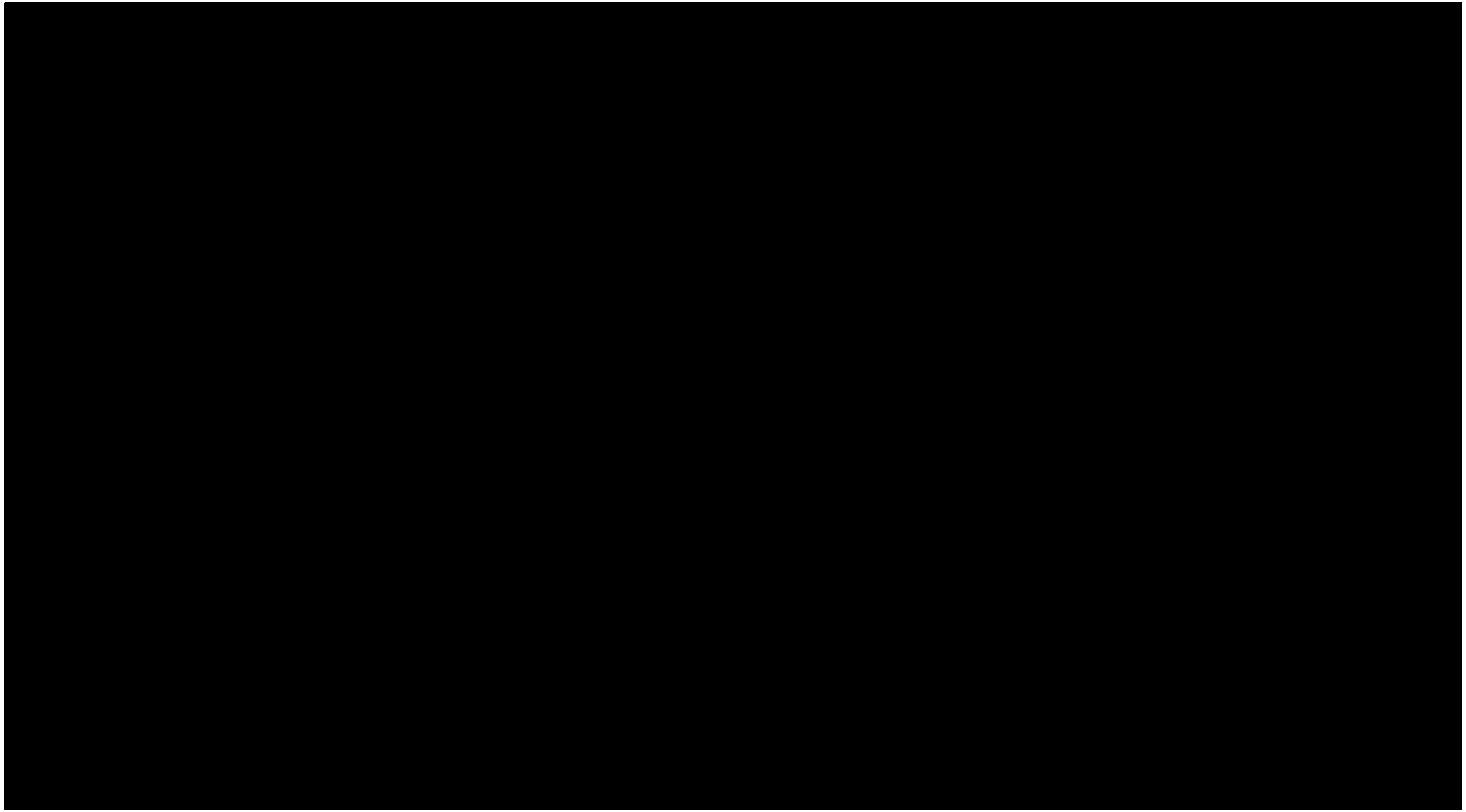


© 2016 Adobe Systems Incorporated. All Rights Reserved.

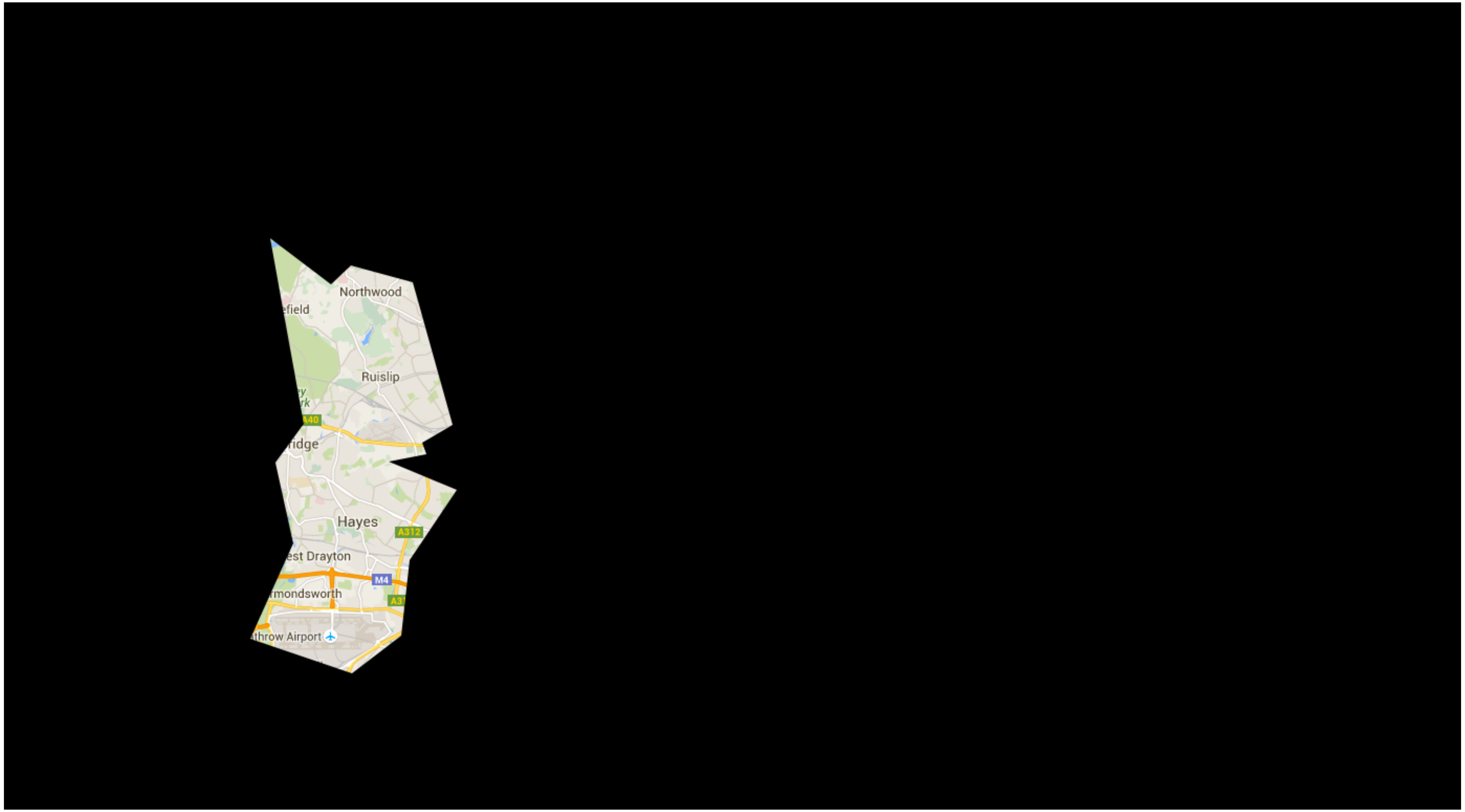


© 2016 Adobe Systems Incorporated. All Rights Reserved.

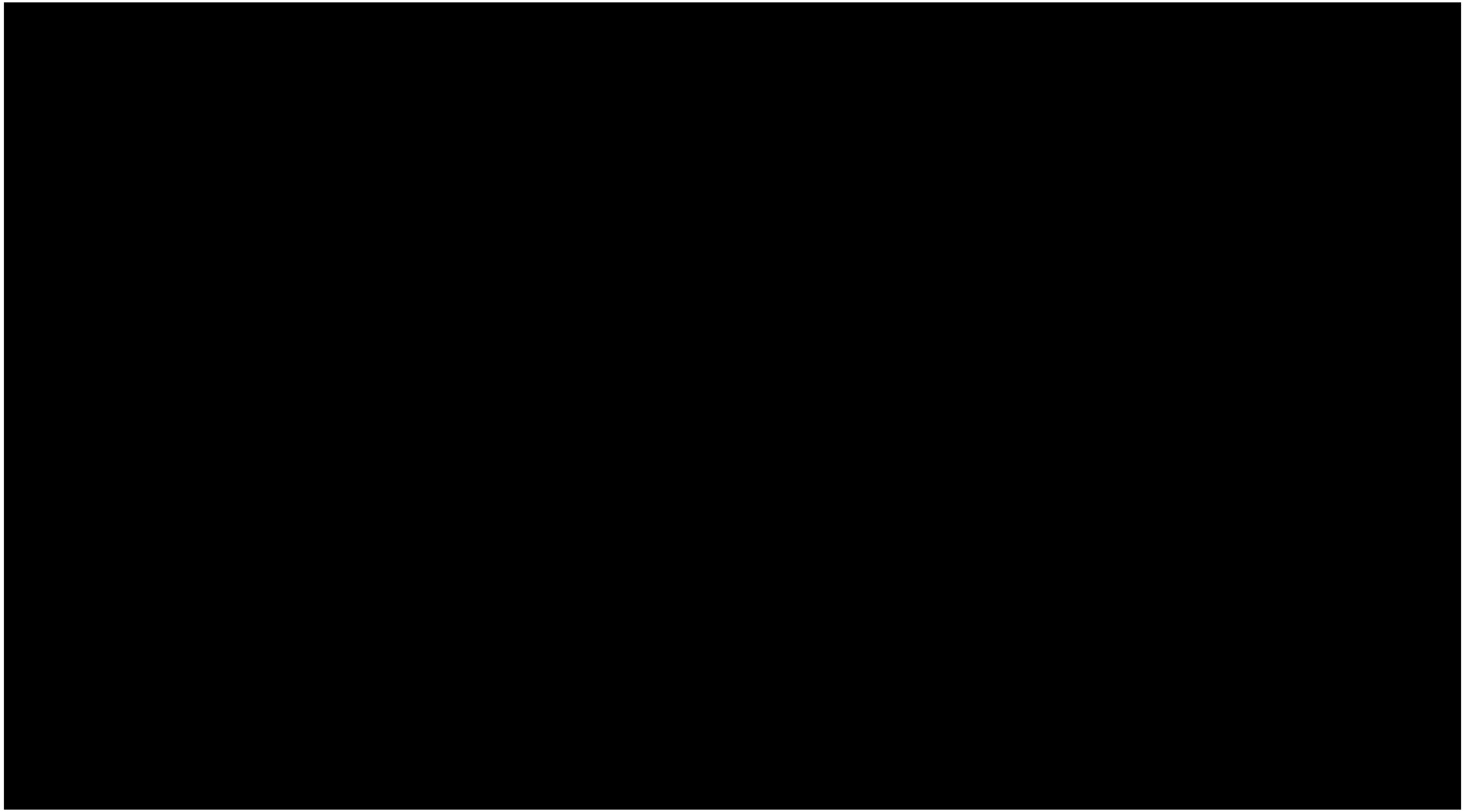

A1
BRENT CROSS
A41
A406



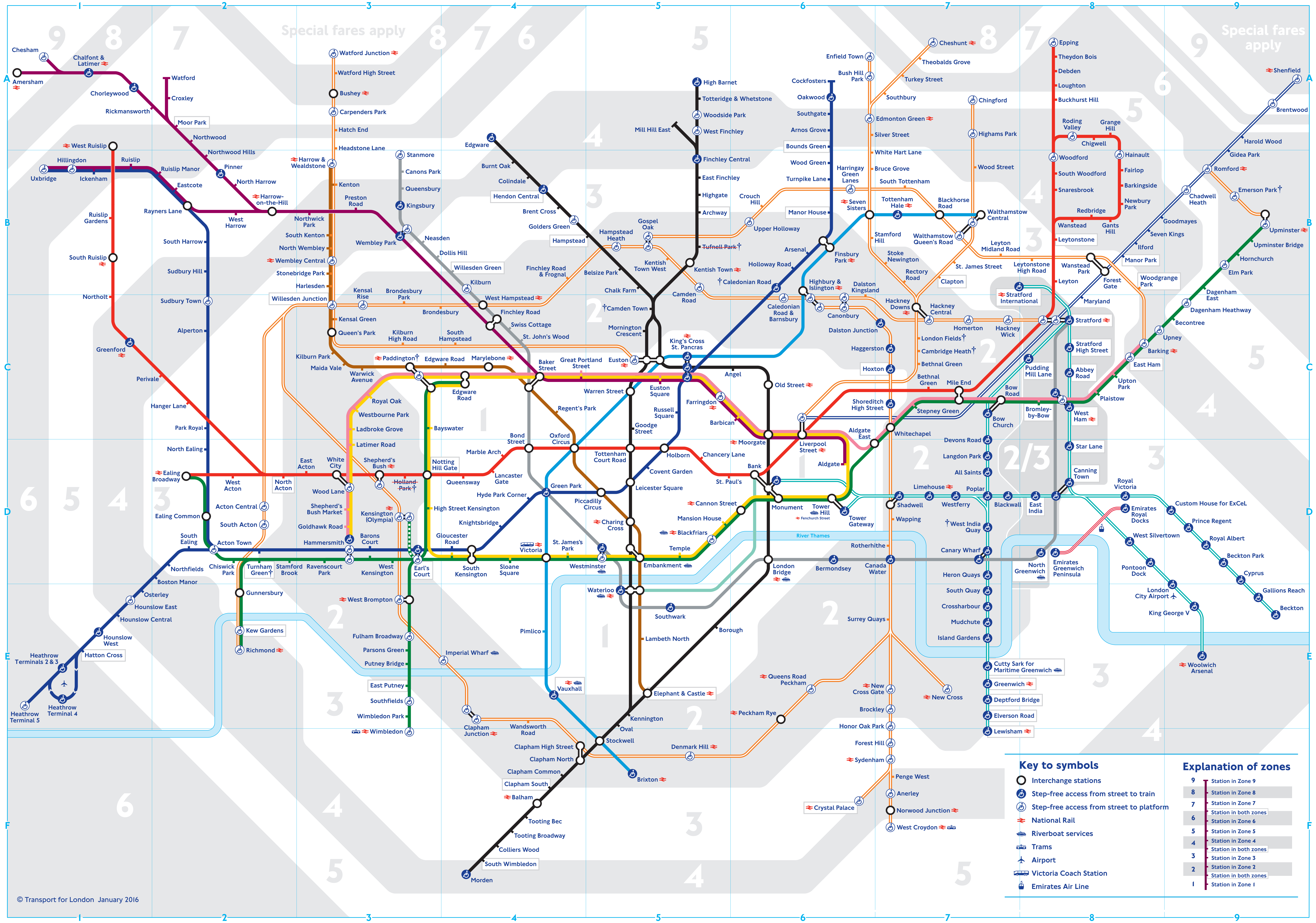
© 2016 Adobe Systems Incorporated. All Rights Reserved.



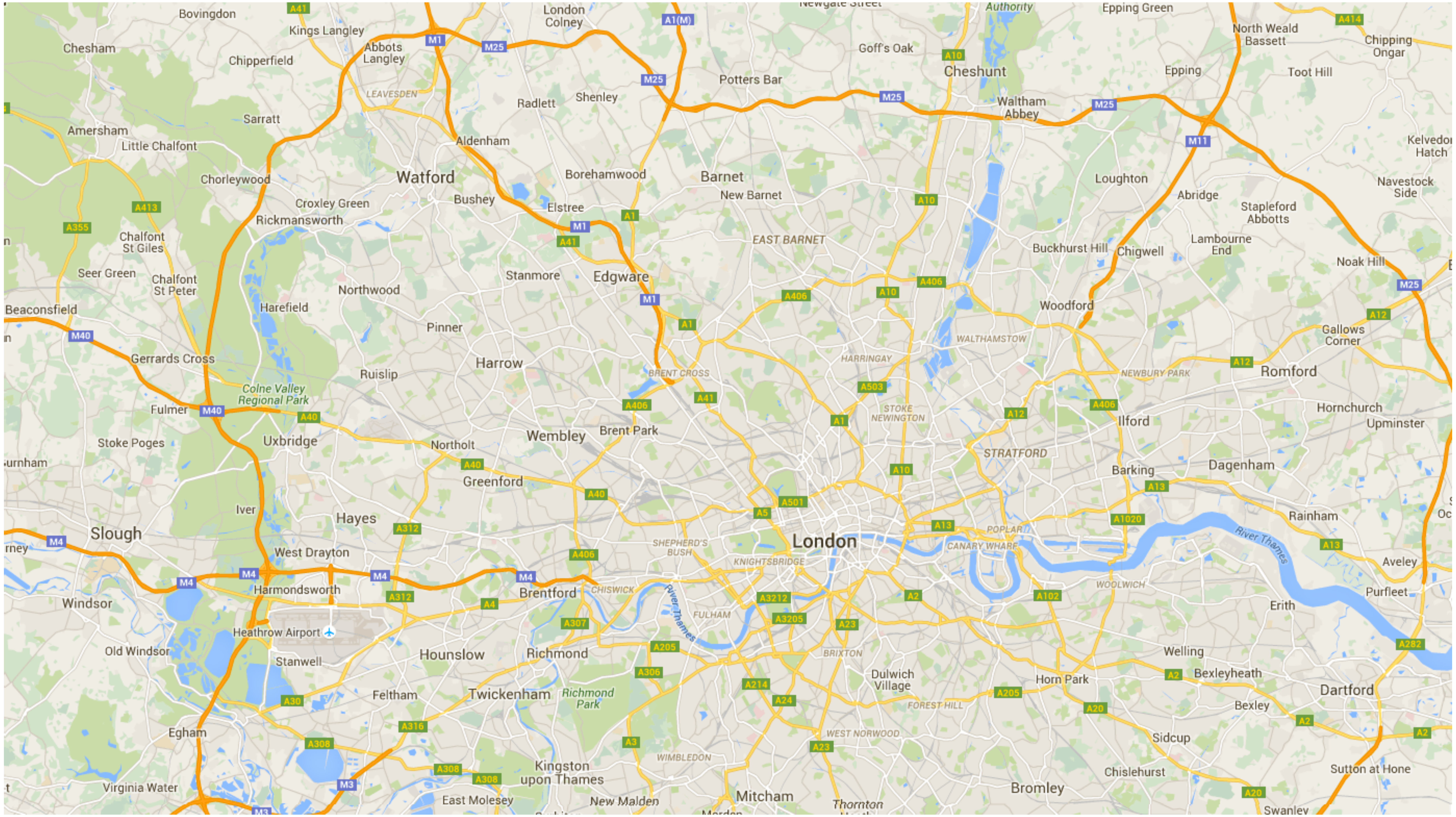
© 2016 Adobe Systems Incorporated. All Rights Reserved.



© 2016 Adobe Systems Incorporated. All Rights Reserved.

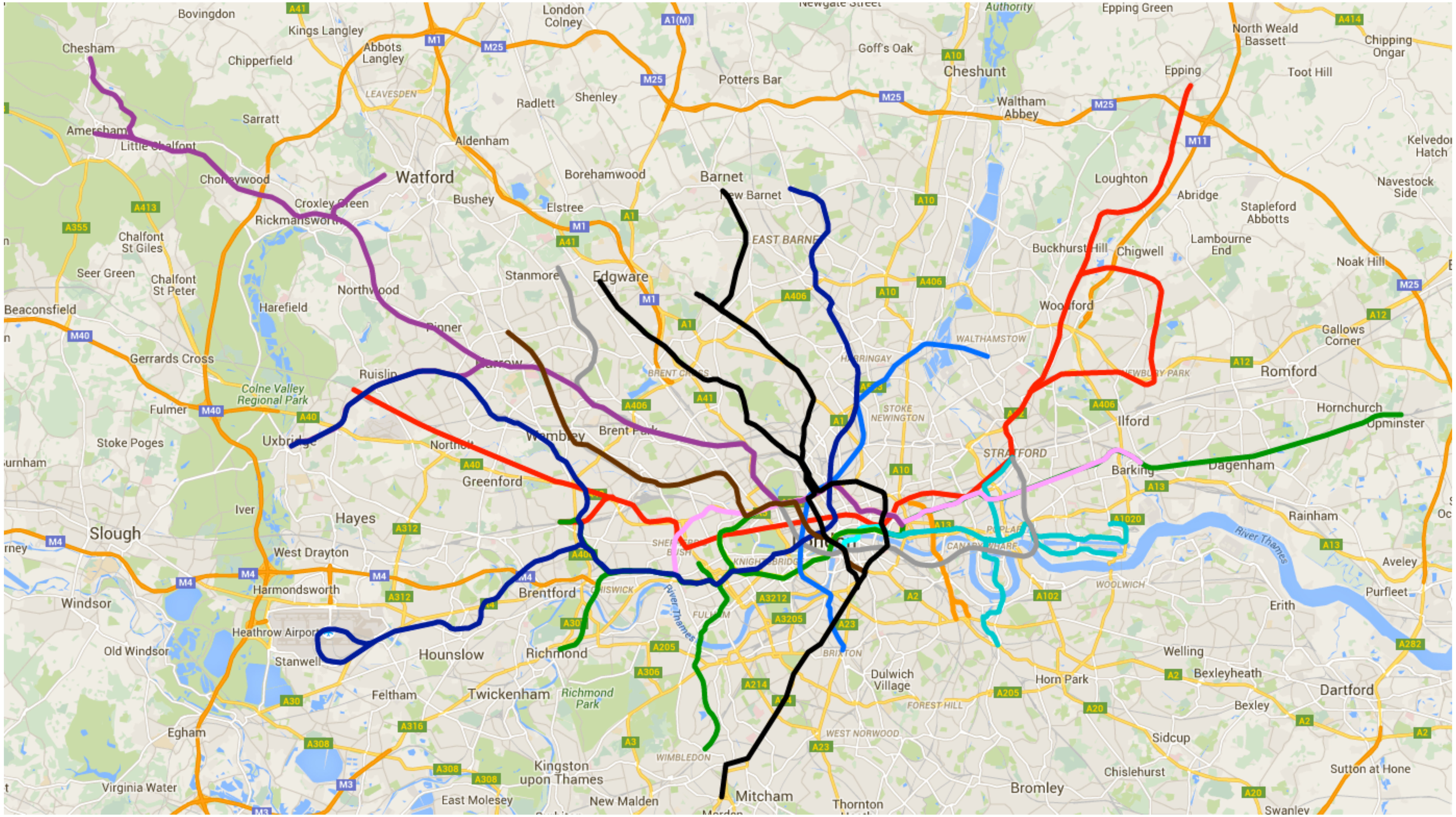


© 2016 Adobe Systems Incorporated. All Rights Reserved.



© 2016 Adobe Systems Incorporated. All Rights Reserved.

Adobe



© 2016 Adobe Systems Incorporated. All Rights Reserved.

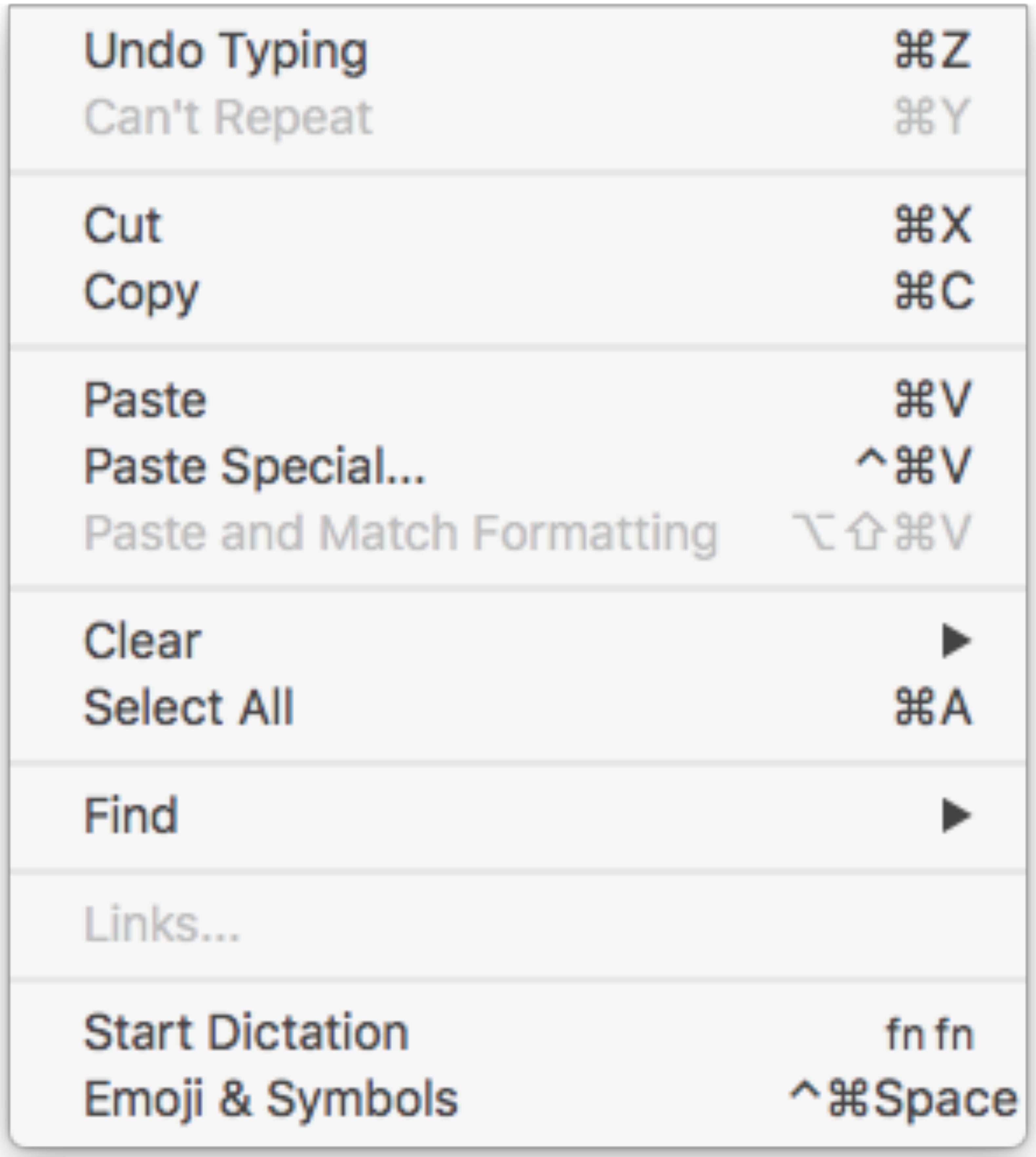
"There are rules!"

– The Big Lebowski

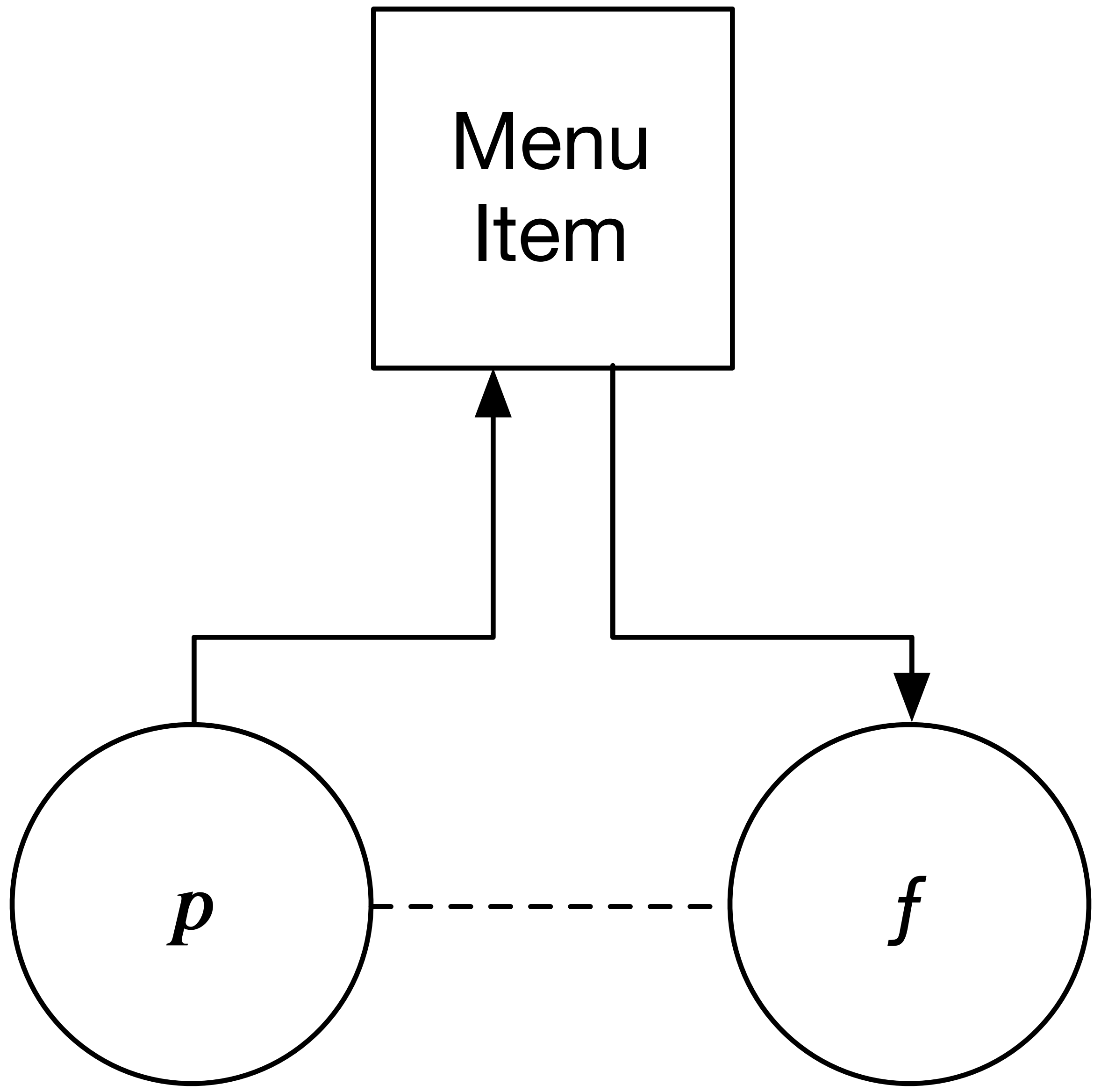
Lower Bound

```
template <class ForwardIterator, class T, class Compare>
ForwardIterator lower_bound(ForwardIterator first, ForwardIterator last,
        const T& value, Compare comp)
{
    auto n = distance(first, last);

    while (n != 0)  {
        auto h = n / 2;
        auto m = next(first, h);

        if (comp(*m, value)) {
            first = next(m);
            n -= h + 1;
        } else { n = h; }
    }

    return first;
}
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

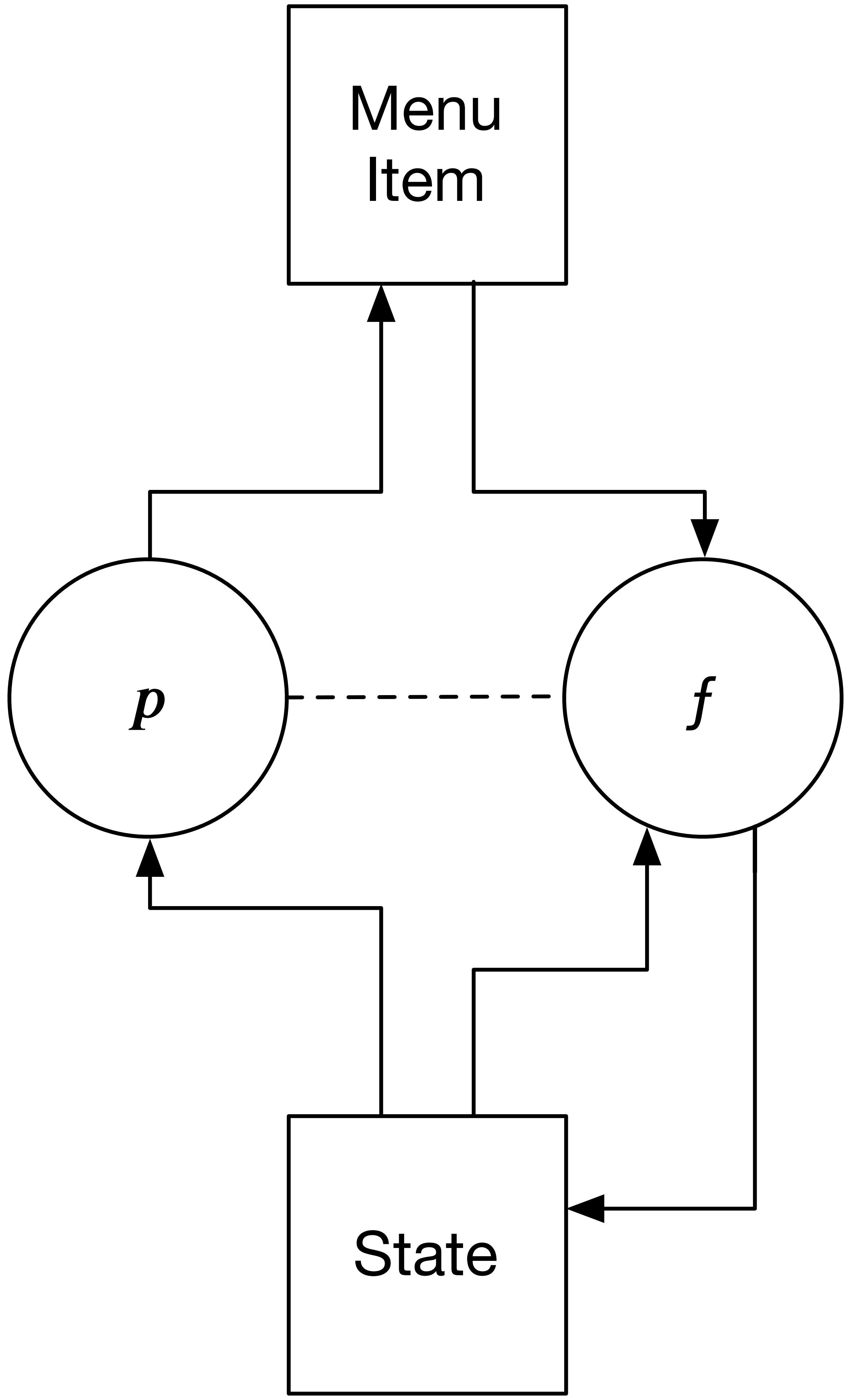


© 2016 Adobe Systems Incorporated. All Rights Reserved.

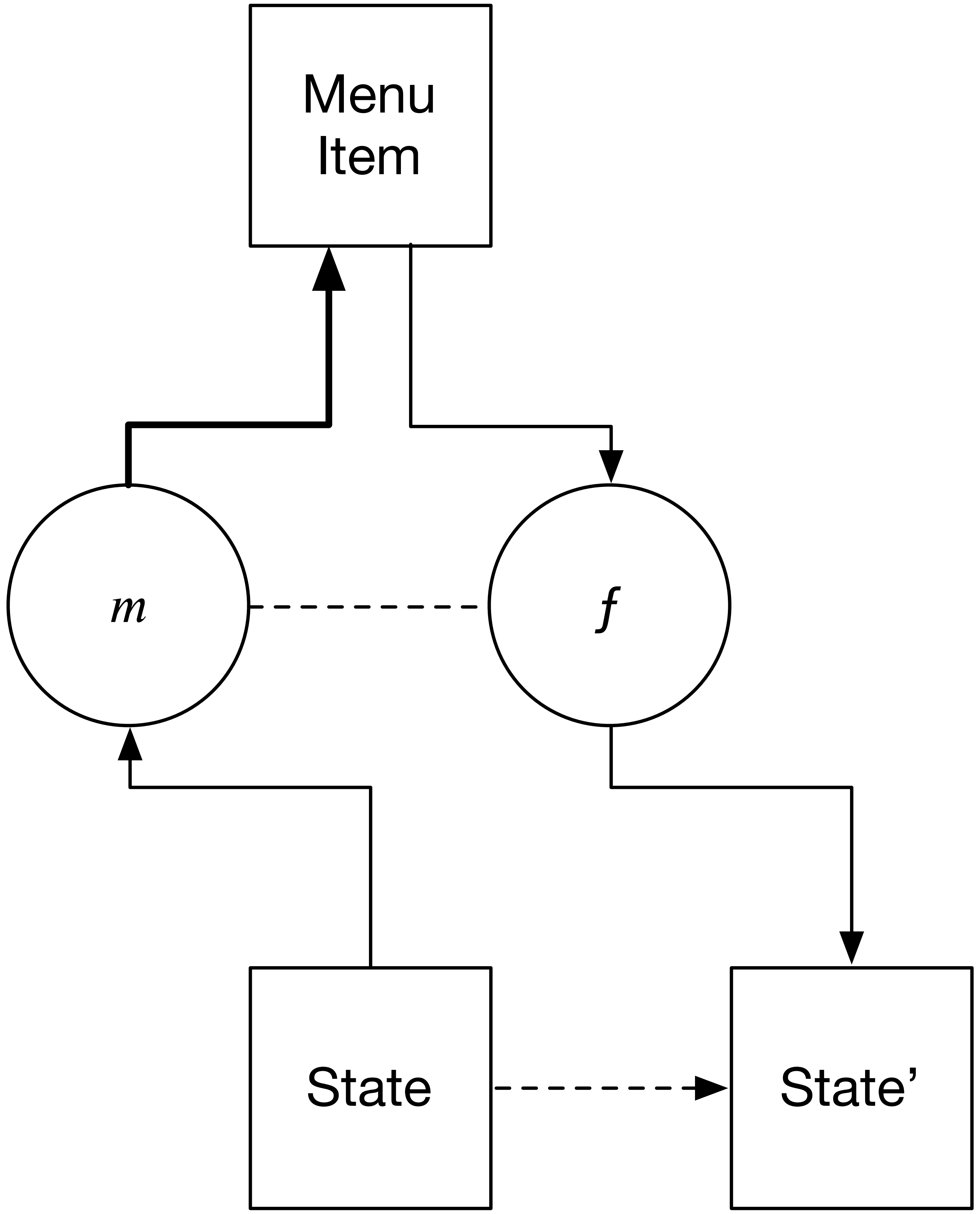


© 2016 Adobe Systems Incorporated. All Rights Reserved.

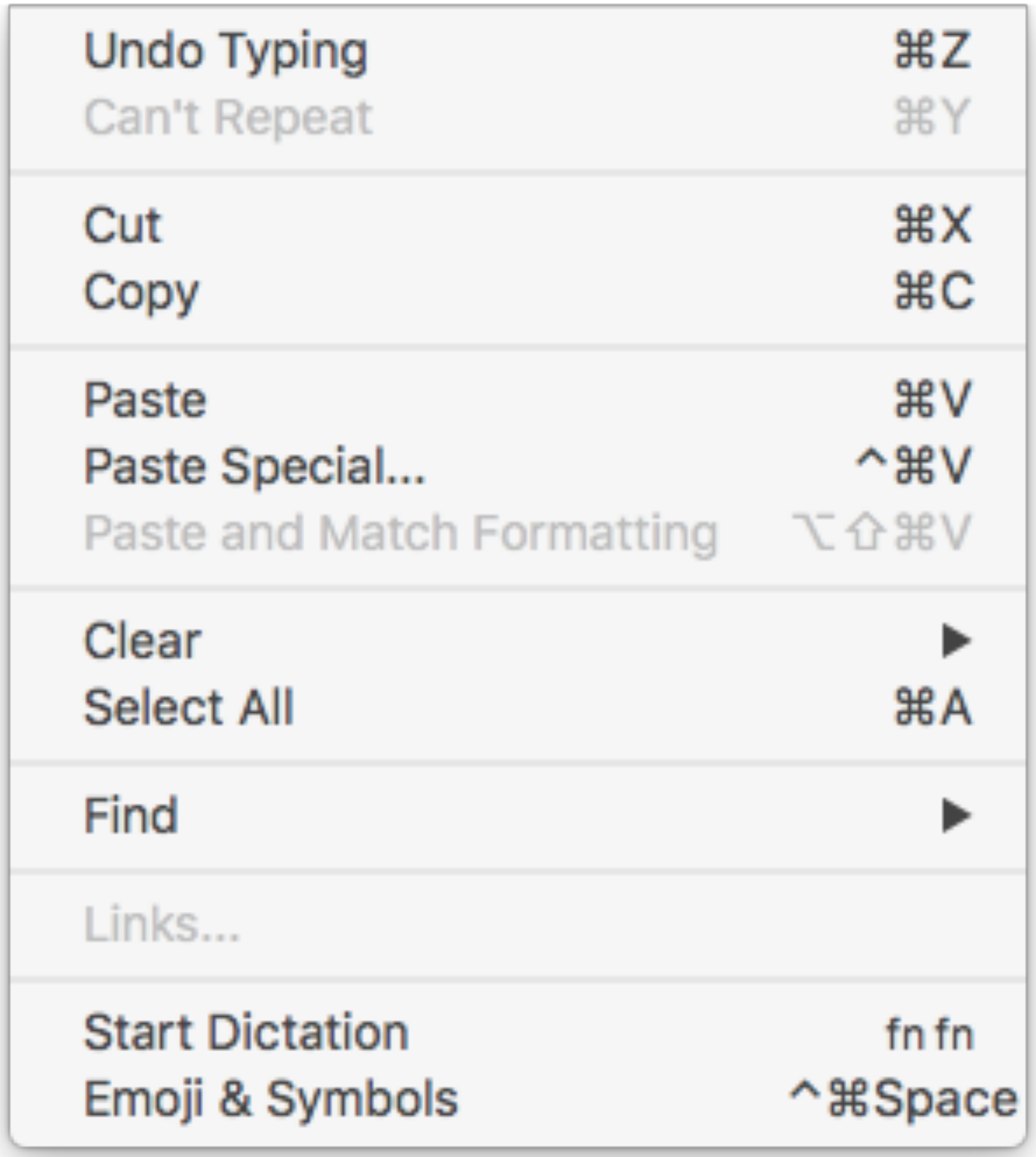
Adobe



© 2016 Adobe Systems Incorporated. All Rights Reserved.



© 2016 Adobe Systems Incorporated. All Rights Reserved.



© 2016 Adobe Systems Incorporated. All Rights Reserved.

Good Code

Good code is *correct*

© 2016 Adobe Systems Incorporated. All Rights Reserved. Adobe Confidential.

Good Code

Good code is *correct*

- Consistent; without contradiction

© 2016 Adobe Systems Incorporated. All Rights Reserved. Adobe Confidential.

Simple Bug

```
void print_string(const char* s) {
    while (*s != '\0') {
        cout << *s++;
    }
}

int main() {
    print_string(nullptr);
}
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Simple Bug

```
void print_string(const char* s) {
    while (*s != '\0') {            Thread 1: EXC_BAD_ACCESS (code=1, address=0x0)
        cout << *s++;
    }
}

int main() {
    print_string(nullptr);
}
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Simple Bug

```
void print_string(const char* s) {
    while (*s != '\0') {
        cout << *s++;
    }
}

int main() {
    print_string(nullptr);
}
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Simple Bug

```
void print_string(const char* s) {
    while (*s != '\0') {
        cout << *s++;
    }
}

int main() {
    print_string(nullptr); // FORCE CRASH!
}
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Subtle defects

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Subtle defects

Consistency requires context

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Subtle defects

Consistency requires context

```
template<class T> const T& min(const T& a, const T& b);
```

Returns: The smaller value.

Remarks: Returns the first argument when the arguments are equivalent.

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Subtle defects

Consistency requires context

```
template<class T> const T& min(const T& a, const T& b);
```
Returns: The smaller value.

Remarks: Returns the first argument when the arguments are equivalent.

```
template<class T> const T& max(const T& a, const T& b);
```
Returns: The larger value.

Remarks: Returns the first argument when the arguments are equivalent.

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Subtle defects

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Subtle defects

```
template<typename T>
const T& clamp(const T& a, const T& lo, const T& hi)
{
    return min(max(lo, a), hi);
}
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Subtle defects

```
template<typename T>
const T& clamp(const T& a, const T& lo, const T& hi)
{
    return min(max(lo, a), hi);
}

template<typename T, typename Compare>
const T& clamp(const T& a, const T& lo, const T& hi, Compare comp)
{
    return min(max(lo, a, comp), hi, comp);
}
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Subtle defects

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Subtle defects

```
int main() {
    using pair = pair<int, string>;

    pair a = { 1, "OK" };

    pair lo = { 1, "FAIL: LO" };
    pair hi = { 2, "FAIL: HI" };

    a = clamp(a, lo, hi, [](const auto& a, const auto& b) {
        return a.first < b.first;
    });

    cout << a.second << endl;
};
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Subtle defects

```
int main() {
    using pair = pair<int, string>;

    pair a = { 1, "OK" };

    pair lo = { 1, "FAIL: LO" };
    pair hi = { 2, "FAIL: HI" };

    a = clamp(a, lo, hi, [](const auto& a, const auto& b) {
        return a.first < b.first;
    });

    cout << a.second << endl;
};
```

FAIL: LO

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Subtle defects

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Subtle defects

```
template<typename T>
const T& clamp(const T& a, const T& lo, const T& hi)
{
    return min(max(a, lo), hi);
}
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Subtle defects

```
template<typename T>
const T& clamp(const T& a, const T& lo, const T& hi)
{
    return min(max(a, lo), hi);
}

template<typename T, typename Compare>
const T& clamp(const T& a, const T& lo, const T& hi, Compare comp)
{
    return min(max(a, lo, comp), hi, comp);
}
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Subtle defects

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Subtle defects

```
template<class T> const T& min(const T& a, const T& b);
```

Returns: The smaller value.

Remarks: Returns the first argument when the arguments are equivalent.

```
template<class T> const T& max(const T& a, const T& b);
```

Returns: The larger value.

Remarks: Returns the **second** argument when the arguments are equivalent.

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Subtle defects

```
template<class T> const T& min(const T& a, const T& b);
```
Returns: The smaller value.

Remarks: Returns the first argument when the arguments are equivalent.

```
template<class T> const T& max(const T& a, const T& b);
```
Returns: The larger value.

Remarks: Returns the **second** argument when the arguments are equivalent.

```
template <class T> const T& max(const T& a, const T& b, const T& c);
```
Returns: The larger value.

Remarks: **???**

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Rules are Contentious

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Rules are Contentious

"Names should not be associated with semantics because everybody has their own hidden assumptions about what semantics are, and they clash, causing comprehension problems without knowing why. This is why it's valuable to write code to reflect what code is actually doing, rather than what code 'means': it's hard to have conceptual clashes about what code actually does."

– Craig Silverstein, Google

© 2016 Adobe Systems Incorporated. All Rights Reserved.


Adobe
“There is no spoon.”
– The Matrix
Bē
GMUNK

How can nothing be something?

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
 // indeterminate value
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
 // indeterminate value

int x = 1 / 0;
```

How can nothing be something?

```
int x;
 // indeterminate value

int x = 1 / 0;
 // undefined behavior
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
 // indeterminate value

int x = 1 / 0;
 // undefined behavior

double x = 1.0 / 0.0;
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
 // indeterminate value

int x = 1 / 0;
 // undefined behavior

double x = 1.0 / 0.0;
 // inf
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
 // indeterminate value

int x = 1 / 0;
 // undefined behavior

double x = 1.0 / 0.0;
 // inf

double x = 0.0 / 0.0;
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
 // indeterminate value

int x = 1 / 0;
 // undefined behavior

double x = 1.0 / 0.0;
 // inf

double x = 0.0 / 0.0;
 // NaN
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
 // indeterminate value

int x = 1 / 0;
 // undefined behavior

double x = 1.0 / 0.0;
 // inf

double x = 0.0 / 0.0;
 // NaN

struct empty { };
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Adobe

How can nothing be something?

```
int x;
 // indeterminate value

int x = 1 / 0;
 // undefined behavior

double x = 1.0 / 0.0;
 // inf

double x = 0.0 / 0.0;
 // NaN

struct empty { };
 // sizeof(empty) == 1
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // Error
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // Error
 // but common extension
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // Error
 // but common extension
 using empty = int[0];
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // Error
 // but common extension
 using empty = int[0];
 // sizeof(empty) == 0
 empty a[2];
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // Error
 // but common extension
 using empty = int[0];
 // sizeof(empty) == 0
 empty a[2];
 // &a[0] == &a[1]
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // Error
 // but common extension
 using empty = int[0];
 // sizeof(empty) == 0
 empty a[2];
 // &a[0] == &a[1]

void f() { return void(); }
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // Error
 // but common extension
 using empty = int[0];
 // sizeof(empty) == 0
 empty a[2];
 // &a[0] == &a[1]

void f() { return void(); }
 // OK
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // Error
 // but common extension
 using empty = int[0];
 // sizeof(empty) == 0
 empty a[2];
 // &a[0] == &a[1]

void f() { return void(); }
 // OK

void x = f();
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // Error
 // but common extension
 using empty = int[0];
 // sizeof(empty) == 0
 empty a[2];
 // &a[0] == &a[1]

void f() { return void(); }
 // OK

void x = f();
 // Error
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // Error
 // but common extension
 using empty = int[0];
 // sizeof(empty) == 0
 empty a[2];
 // &a[0] == &a[1]

void f() { return void(); }
 // OK

void x = f();
 // Error
 // but void* is a pointer to anything…
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
std::vector<int> x = { 1, 2, 3 };
try {
    x.insert(x.begin(), 0);
} catch (...) {
    std::cout << x.size() << std::endl;
}
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
std::vector<int> x = { 1, 2, 3 };
try {
    x.insert(x.begin(), 0);
} catch (...) {
    std::cout << x.size() << std::endl;
}
 // Basic Exception Guarantee:
 // Valid but unspecified
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
std::vector<int> x = { 1, 2, 3 };
try {
    x.insert(x.begin(), 0);
} catch (...) {
    std::cout << x.size() << std::endl;
}
 // Basic Exception Guarantee:
 // Valid but unspecified

std::vector<int> y = std::move(x);
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
std::vector<int> x = { 1, 2, 3 };
try {
    x.insert(x.begin(), 0);
} catch (...) {
    std::cout << x.size() << std::endl;
}
 // Basic Exception Guarantee:
 // Valid but unspecified

std::vector<int> y = std::move(x);
 // Moved from object, x, is valid but unspecified
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Good Code

© 2016 Adobe Systems Incorporated. All Rights Reserved. Adobe Confidential.

Good Code

Good code is *correct*

- Consistent; without contradiction

© 2016 Adobe Systems Incorporated. All Rights Reserved. Adobe Confidential.

Good Code

Good code is *correct*

- Consistent; without contradiction

Good code has *meaning*

© 2016 Adobe Systems Incorporated. All Rights Reserved. Adobe Confidential.

Good Code

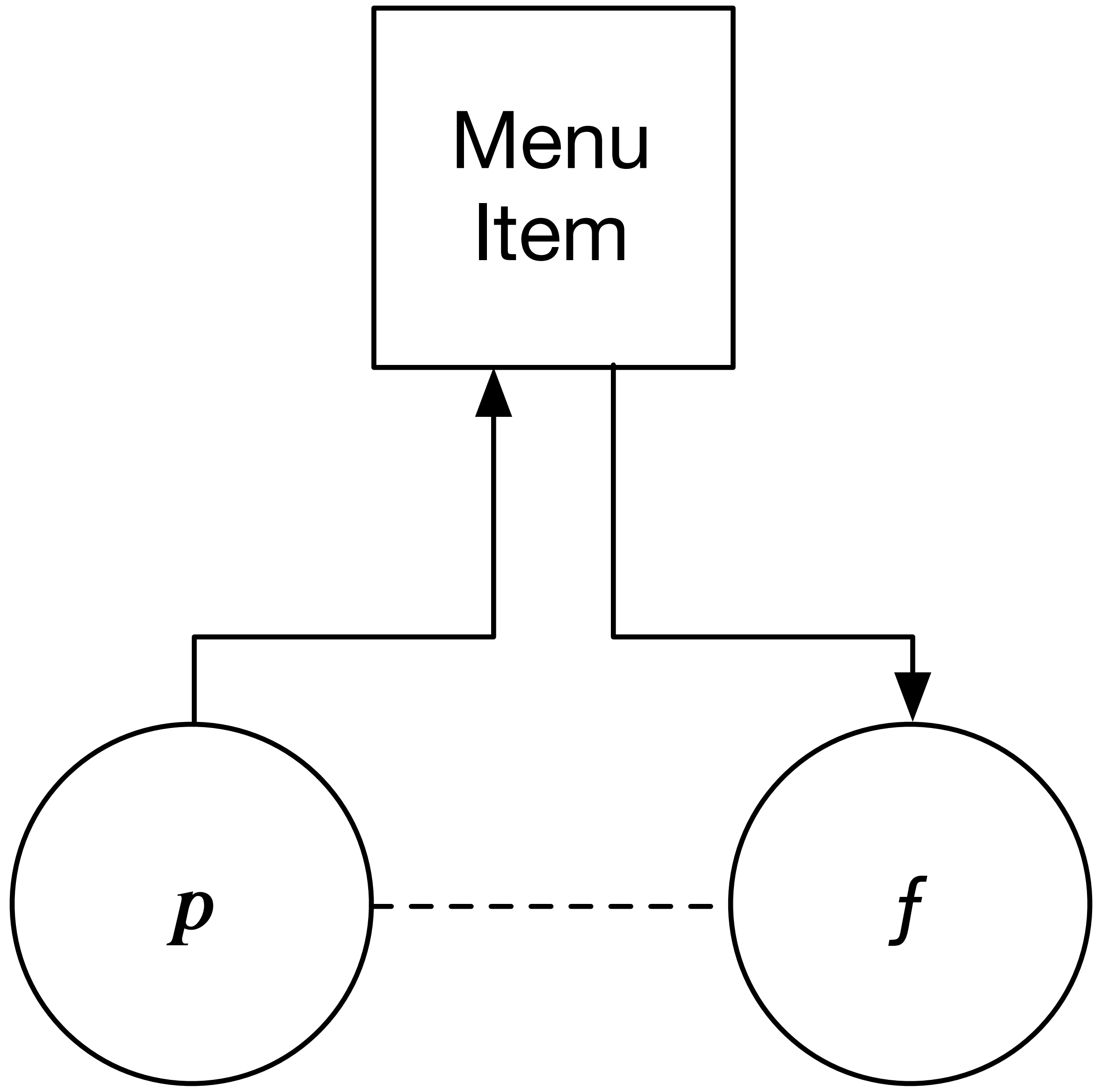
Good code is *correct*

Consistent; without contradiction

Good code has *meaning*

Correspondence to an entity; specified, defined

© 2016 Adobe Systems Incorporated. All Rights Reserved. Adobe Confidential.



© 2016 Adobe Systems Incorporated. All Rights Reserved.

Categories of nothing

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Categories of nothing

Absence of *something*

0, ∅, [p, p), void

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Categories of nothing

Absence of *something*

  0, Ø, [p, p), void

Absence of *meaning*

  NaN, undefined, indeterminate

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
 // Partially formed; assign value or destruct
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
 // Partially formed; assign value or destruct

int x = 1 / 0;
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
 // Partially formed; assign value or destruct

int x = 1 / 0;
 // undefined behavior; reading from meaningless value
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
 // Partially formed; assign value or destruct

int x = 1 / 0;
 // undefined behavior; reading from meaningless value

double x = 1.0 / 0.0;
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
 // Partially formed; assign value or destruct

int x = 1 / 0;
 // undefined behavior; reading from meaningless value

double x = 1.0 / 0.0;
 // inf; OK, approximation for underflow
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
 // Partially formed; assign value or destruct

int x = 1 / 0;
 // undefined behavior; reading from meaningless value

double x = 1.0 / 0.0;
 // inf; OK, approximation for underflow

double x = 0.0 / 0.0;
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
 // Partially formed; assign value or destruct

int x = 1 / 0;
 // undefined behavior; reading from meaningless value

double x = 1.0 / 0.0;
 // inf; OK, approximation for underflow

double x = 0.0 / 0.0;
 // NaN; OK, though undefined behavior would also be
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
 // Partially formed; assign value or destruct

int x = 1 / 0;
 // undefined behavior; reading from meaningless value

double x = 1.0 / 0.0;
 // inf; OK, approximation for underflow

double x = 0.0 / 0.0;
 // NaN; OK, though undefined behavior would also be
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
 // Partially formed; assign value or destruct

int x = 1 / 0;
 // undefined behavior; reading from meaningless value

double x = 1.0 / 0.0;
 // inf; OK, approximation for underflow

double x = 0.0 / 0.0;
 // NaN; OK, though undefined behavior would also be

struct empty : void { };
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int x;
 // Partially formed; assign value or destruct

int x = 1 / 0;
 // undefined behavior; reading from meaningless value

double x = 1.0 / 0.0;
 // inf; OK, approximation for underflow

double x = 0.0 / 0.0;
 // NaN; OK, though undefined behavior would also be

struct empty : void { };
 // sizeof(empty) == 0;
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // OK
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // OK
 using empty = int[0];
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // OK
 using empty = int[0];
 // sizeof(empty) == 0
 empty a[2];
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // OK
 using empty = int[0];
 // sizeof(empty) == 0
 empty a[2];
 // &a[0] == &a[1]
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // OK
 using empty = int[0];
 // sizeof(empty) == 0
 empty a[2];
 // &a[0] == &a[1]

void f() { return void(); }
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // OK
 using empty = int[0];
 // sizeof(empty) == 0
 empty a[2];
 // &a[0] == &a[1]

void f() { return void(); }
 // OK
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // OK
 using empty = int[0];
 // sizeof(empty) == 0
 empty a[2];
 // &a[0] == &a[1]

void f() { return void(); }
 // OK

void x = f();
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
int a[0];
 // OK
 using empty = int[0];
 // sizeof(empty) == 0
 empty a[2];
 // &a[0] == &a[1]

void f() { return void(); }
 // OK

void x = f();
 // OK
 // void* is OK
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
std::vector<int> x = { 1, 2, 3 };
try {
    x.insert(x.begin(), 0);
} catch (...) {
    std::cout << x.size() << std::endl;
}
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
std::vector<int> x = { 1, 2, 3 };
try {
    x.insert(x.begin(), 0);
} catch (...) {
    std::cout << x.size() << std::endl;
}
 // Basic Exception Guarantee:
 // Partially formed object. Reading is undefined behavior
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
std::vector<int> x = { 1, 2, 3 };
try {
    x.insert(x.begin(), 0);
} catch (...) {
    std::cout << x.size() << std::endl;
}
 // Basic Exception Guarantee:
 // Partially formed object. Reading is undefined behavior

std::vector<int> y = std::move(x);
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

How can nothing be something?

```
std::vector<int> x = { 1, 2, 3 };
try {
    x.insert(x.begin(), 0);
} catch (...) {
    std::cout << x.size() << std::endl;
}
 // Basic Exception Guarantee:
 // Partially formed object. Reading is undefined behavior

std::vector<int> y = std::move(x);
 // Moved from object, x, is partially formed
```

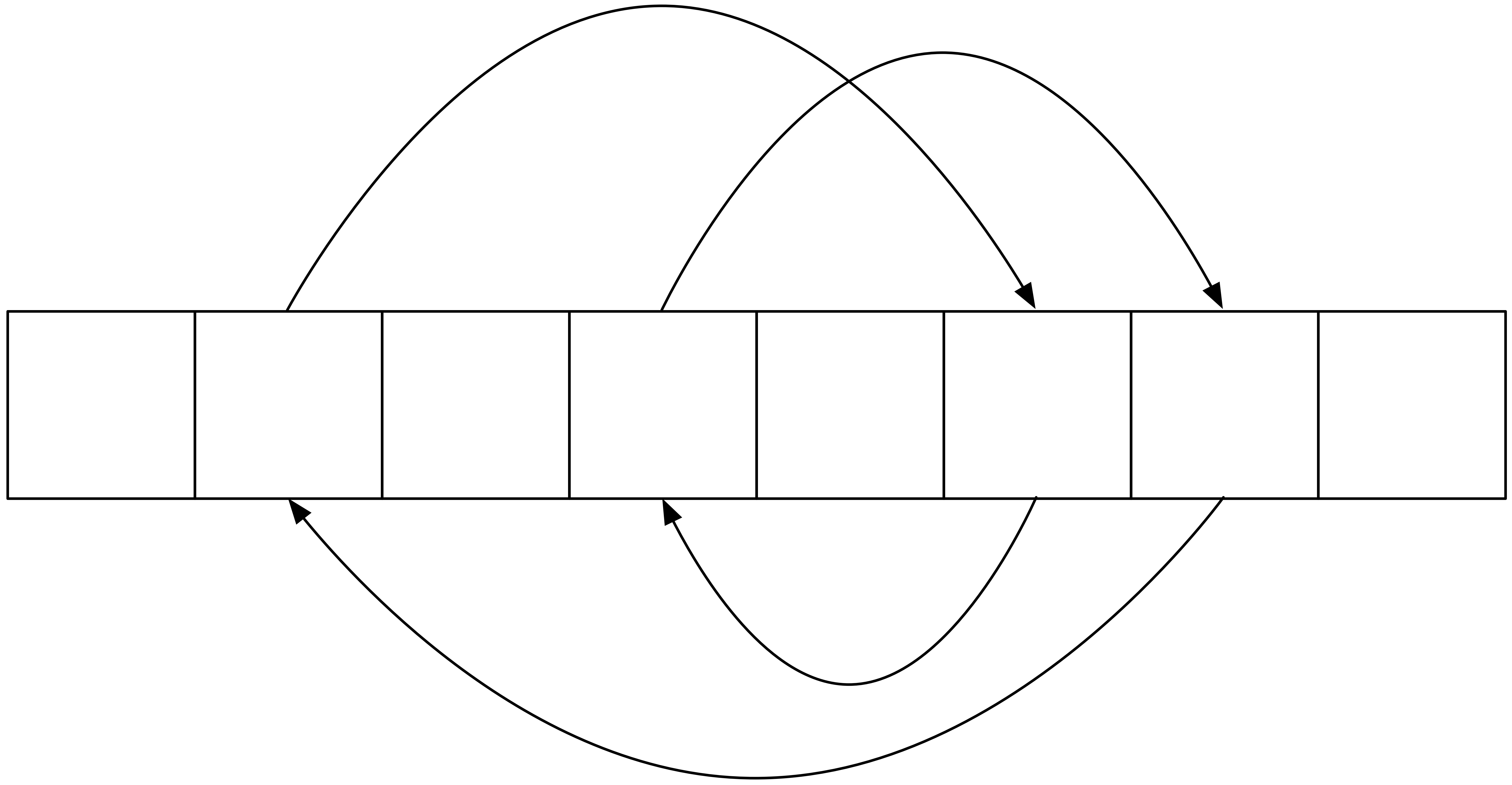
© 2016 Adobe Systems Incorporated. All Rights Reserved.

Adobe

"What's in the box?"

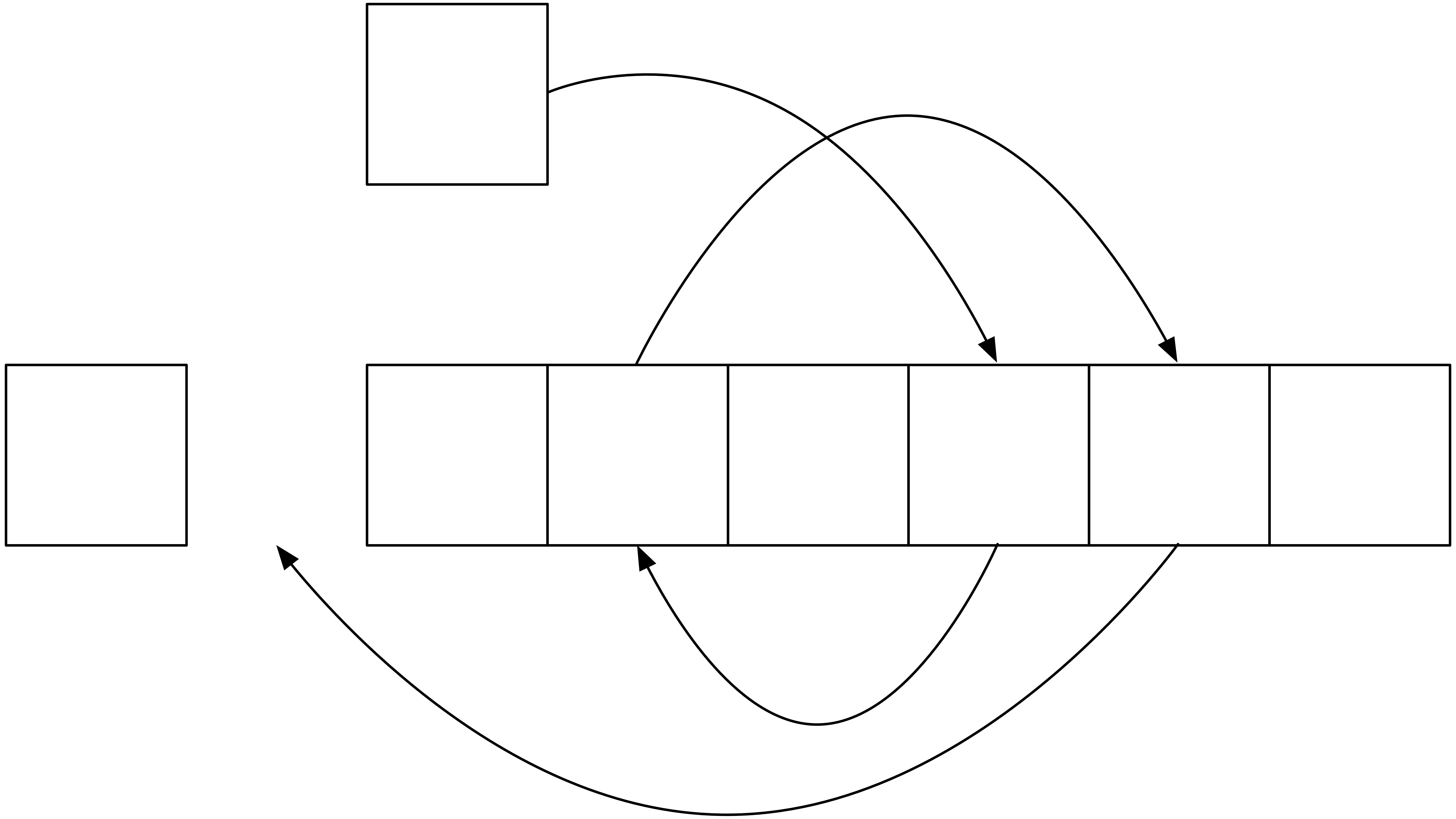
– Seven

The Permutation Paradox



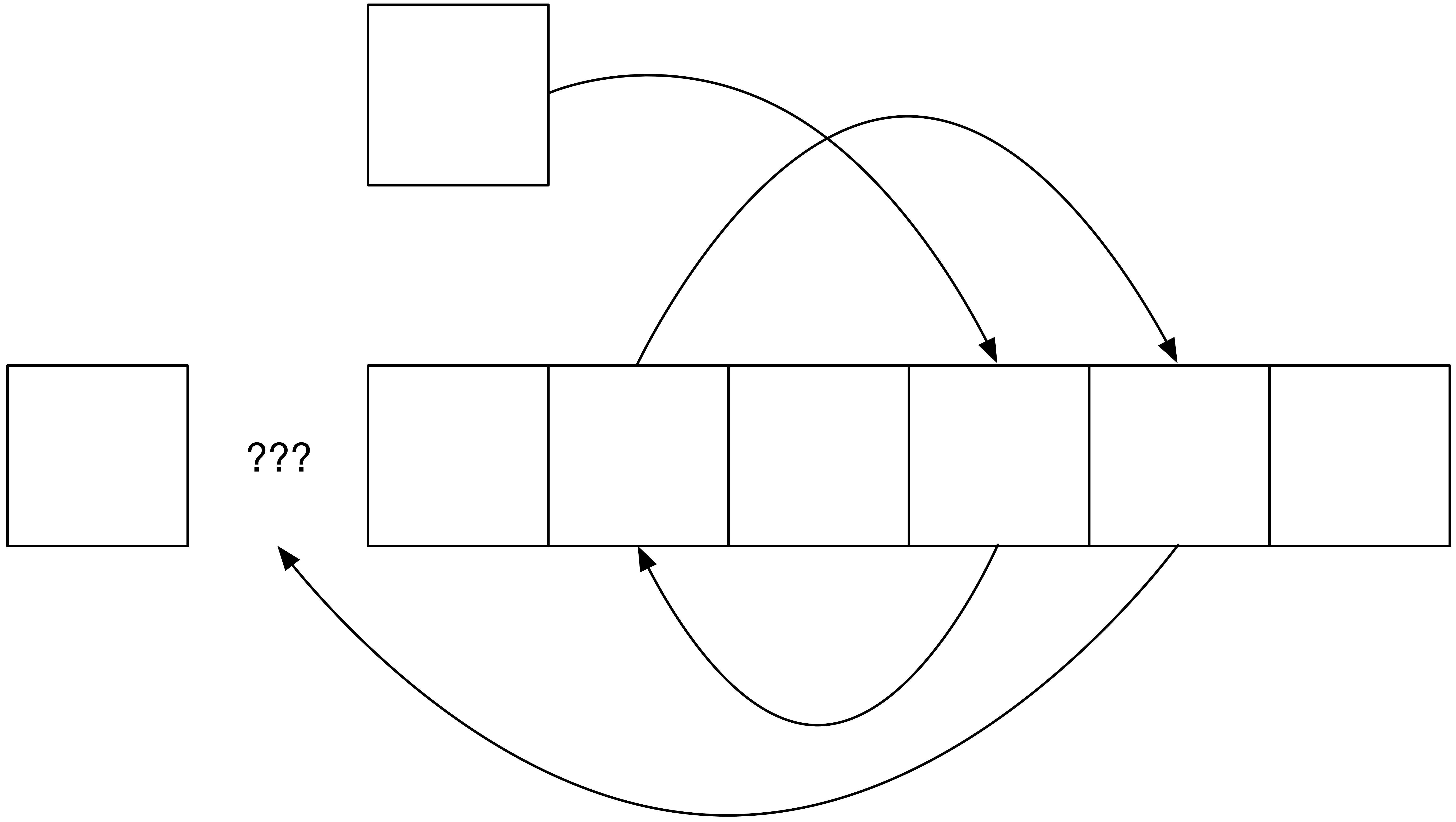
© 2016 Adobe Systems Incorporated. All Rights Reserved.

The Permutation Paradox



© 2016 Adobe Systems Incorporated. All Rights Reserved.

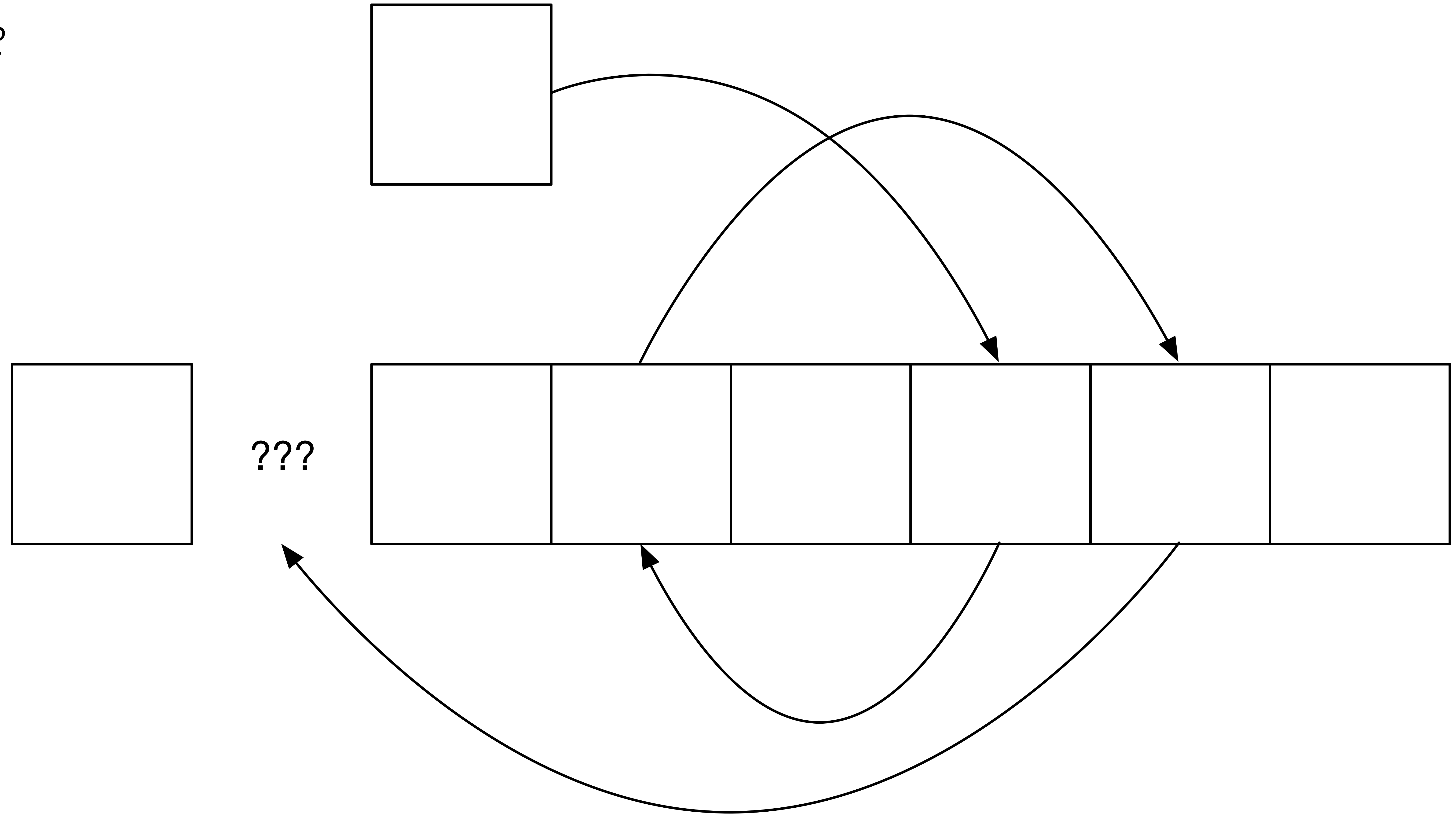
The Permutation Paradox



© 2016 Adobe Systems Incorporated. All Rights Reserved.

The Permutation Paradox

nothing ⇒ unsafe

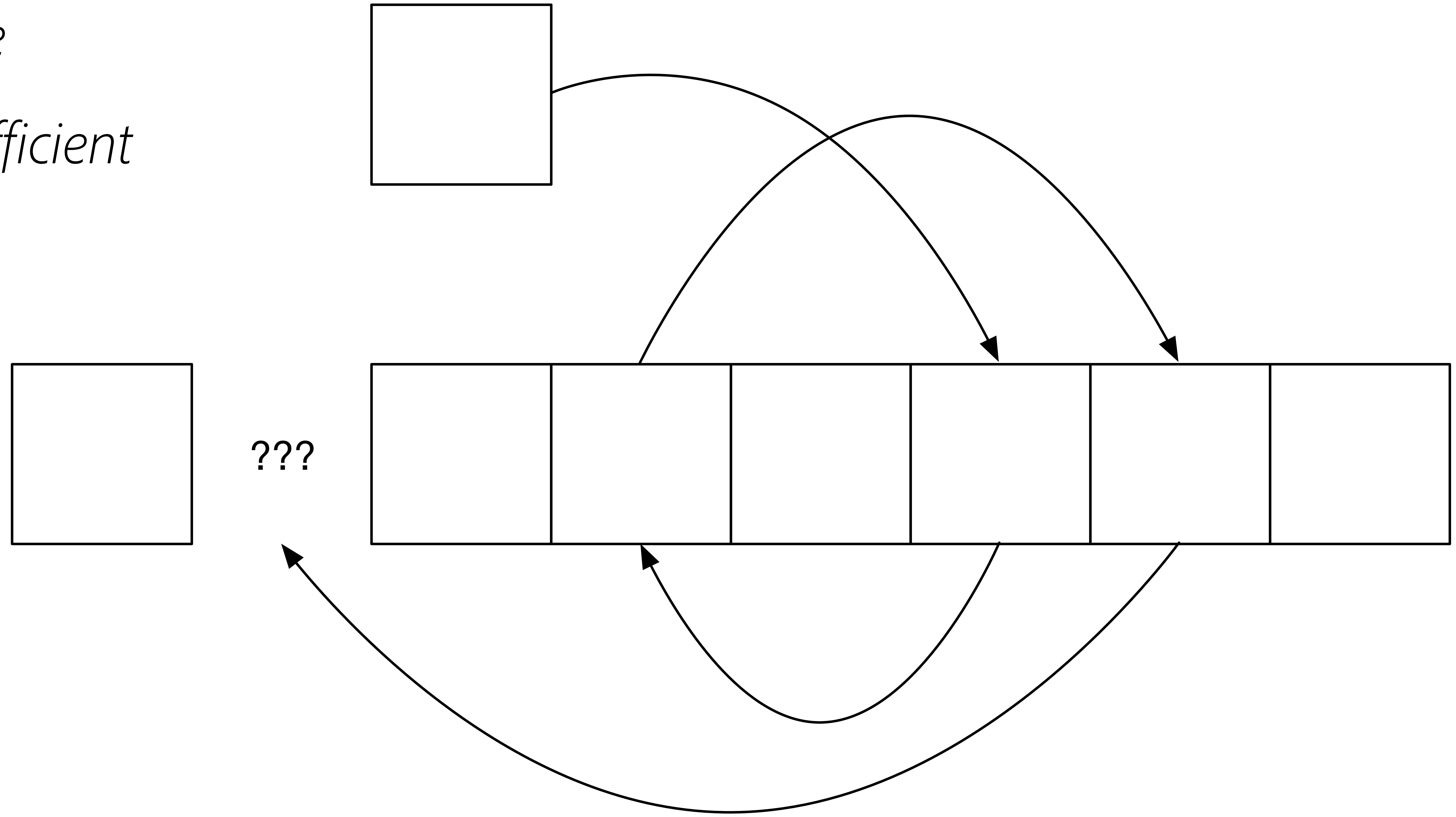


© 2016 Adobe Systems Incorporated. All Rights Reserved.

The Permutation Paradox

nothing ⇒ *unsafe*

something ⇒ *inefficient*



© 2016 Adobe Systems Incorporated. All Rights Reserved.

The Permutation Paradox

© 2016 Adobe Systems Incorporated. All Rights Reserved.

The Permutation Paradox

"There is a duality between transformations and the corresponding actions: An action is definable in terms of a transformation and vice versa:

© 2016 Adobe Systems Incorporated. All Rights Reserved.

The Permutation Paradox

"There is a duality between transformations and the corresponding actions: An action is definable in terms of a transformation and vice versa:

```
void a(T& x) { x = f(x); } // action from transformation
```

and

```
T f(T x) { a(x); return x; } // transformation from action
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

The Permutation Paradox

"There is a duality between transformations and the corresponding actions: An action is definable in terms of a transformation and vice versa:

```
void a(T& x) { x = f(x); } // action from transformation
```

and

```
T f(T x) { a(x); return x; } // transformation from action
```

Despite this duality, independent implementations are sometimes more efficient, in which case both action and transformation need to be provided."

– *Elements of Programming (section 2.5)*

© 2016 Adobe Systems Incorporated. All Rights Reserved.

"It's not that I'm lazy, it's that I just don't care."

– Office Space

Good Code

© 2016 Adobe Systems Incorporated. All Rights Reserved. Adobe Confidential.

Good Code

Good code is *correct*

Consistent; without contradiction

Good code has *meaning*

Correspondence to an entity; specified, defined

© 2016 Adobe Systems Incorporated. All Rights Reserved. Adobe Confidential.

Good Code

Good code is *correct*

Consistent; without contradiction

Good code has *meaning*

Correspondence to an entity; specified, defined

Good code is *efficient*

© 2016 Adobe Systems Incorporated. All Rights Reserved. Adobe Confidential.

Good Code

Good code is *correct*

Consistent; without contradiction

Good code has *meaning*

Correspondence to an entity; specified, defined

Good code is *efficient*

Maximum effect with minimum resources

© 2016 Adobe Systems Incorporated. All Rights Reserved. Adobe Confidential.

Efficiency

© 2016 Adobe Systems Incorporated. All Rights Reserved.

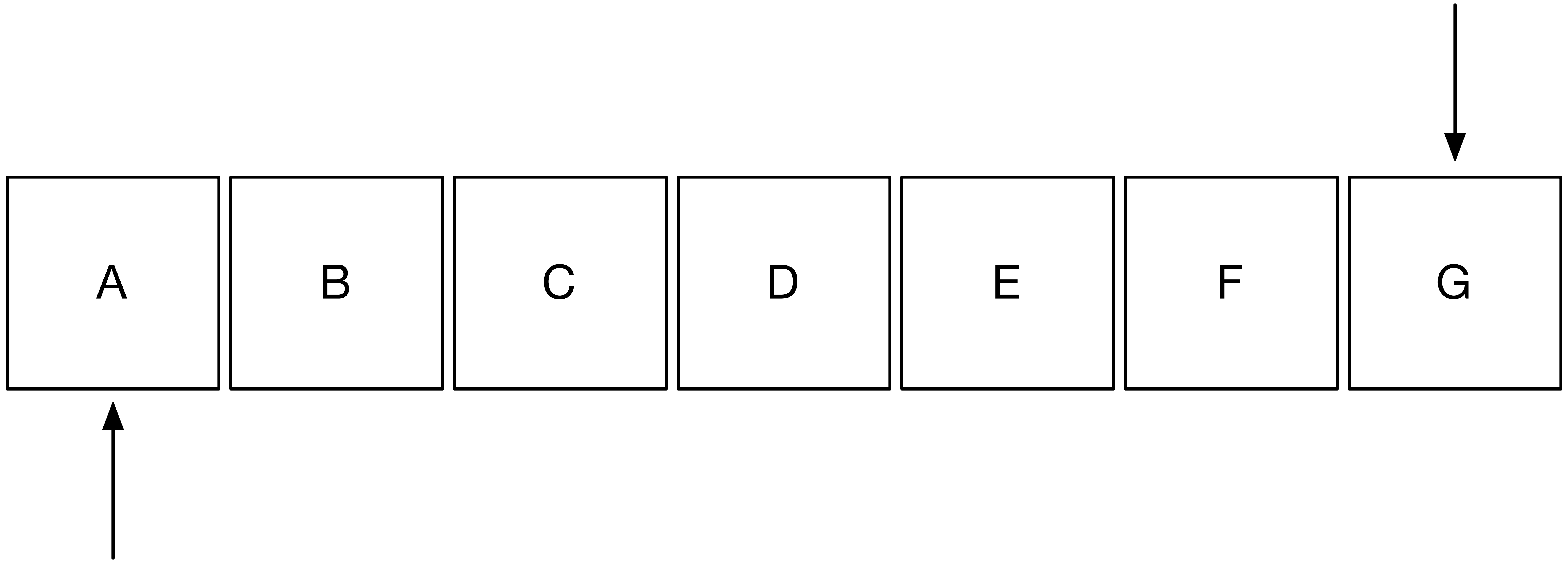
Efficiency

Choice of data structures and algorithms

Choice of what to optimize for

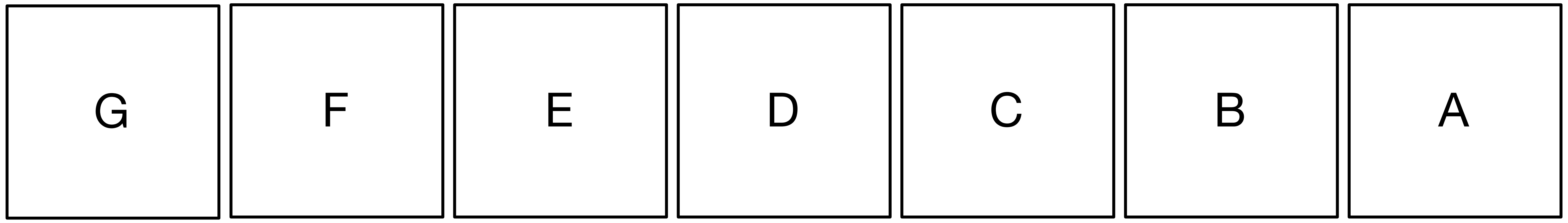
© 2016 Adobe Systems Incorporated. All Rights Reserved.

Efficiency



© 2016 Adobe Systems Incorporated. All Rights Reserved.

Efficiency



© 2016 Adobe Systems Incorporated. All Rights Reserved.

Efficiency

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Efficiency

```
template <class ForwardIterator>
void reverse(ForwardIterator f, ForwardIterator l) {
    auto n = distance(f, l);

    if (n == 0 || n == 1) return;

    auto m = next(f, n / 2);

    reverse(f, m);
    reverse(m, l);
    rotate(f, m, l);
}
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

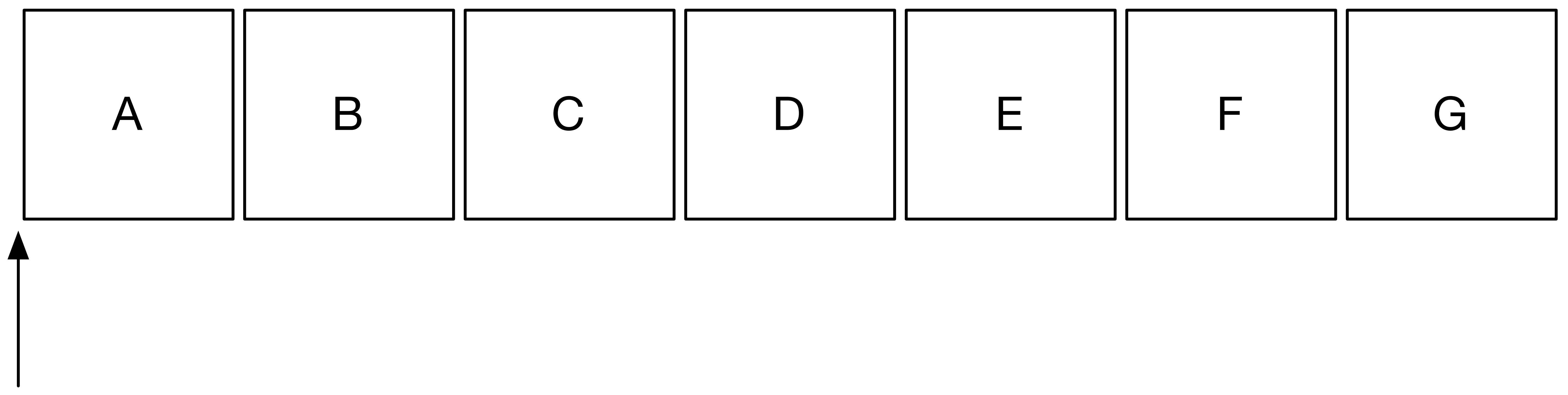
Efficiency

```
template <class ForwardIterator>
void reverse(ForwardIterator f, ForwardIterator l) {
    auto n = distance(f, l);

    if (n == 0 || n == 1) return;

    auto m = next(f, n / 2);

    reverse(f, m);
    reverse(m, l);
    rotate(f, m, l);
}
```

O(n log n)

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Efficiency



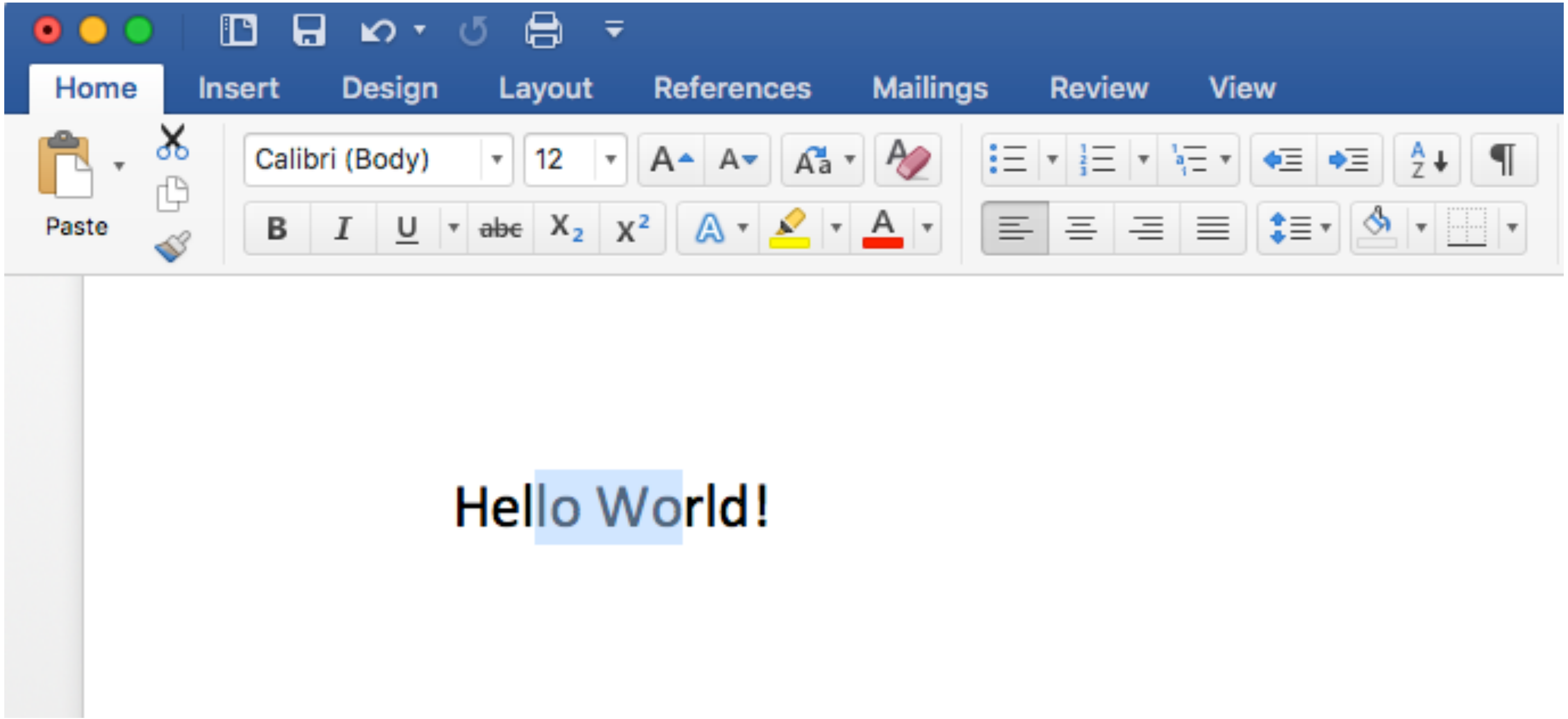
© 2016 Adobe Systems Incorporated. All Rights Reserved.

Efficiency

G	F	E	D	C	B	A

© 2016 Adobe Systems Incorporated. All Rights Reserved.

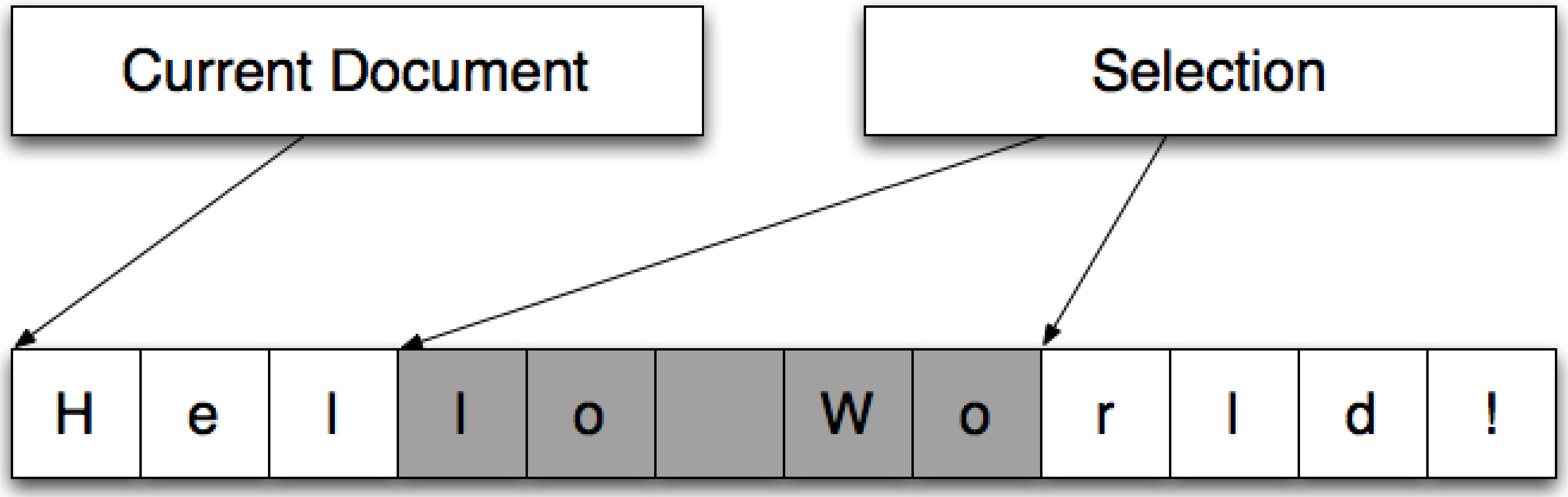
Simple Word Model



© 2016 Adobe Systems Incorporated. All Rights Reserved.

Simple Word Model

- Current Document
- Selection
  - Provides a range; an empty range denotes a location

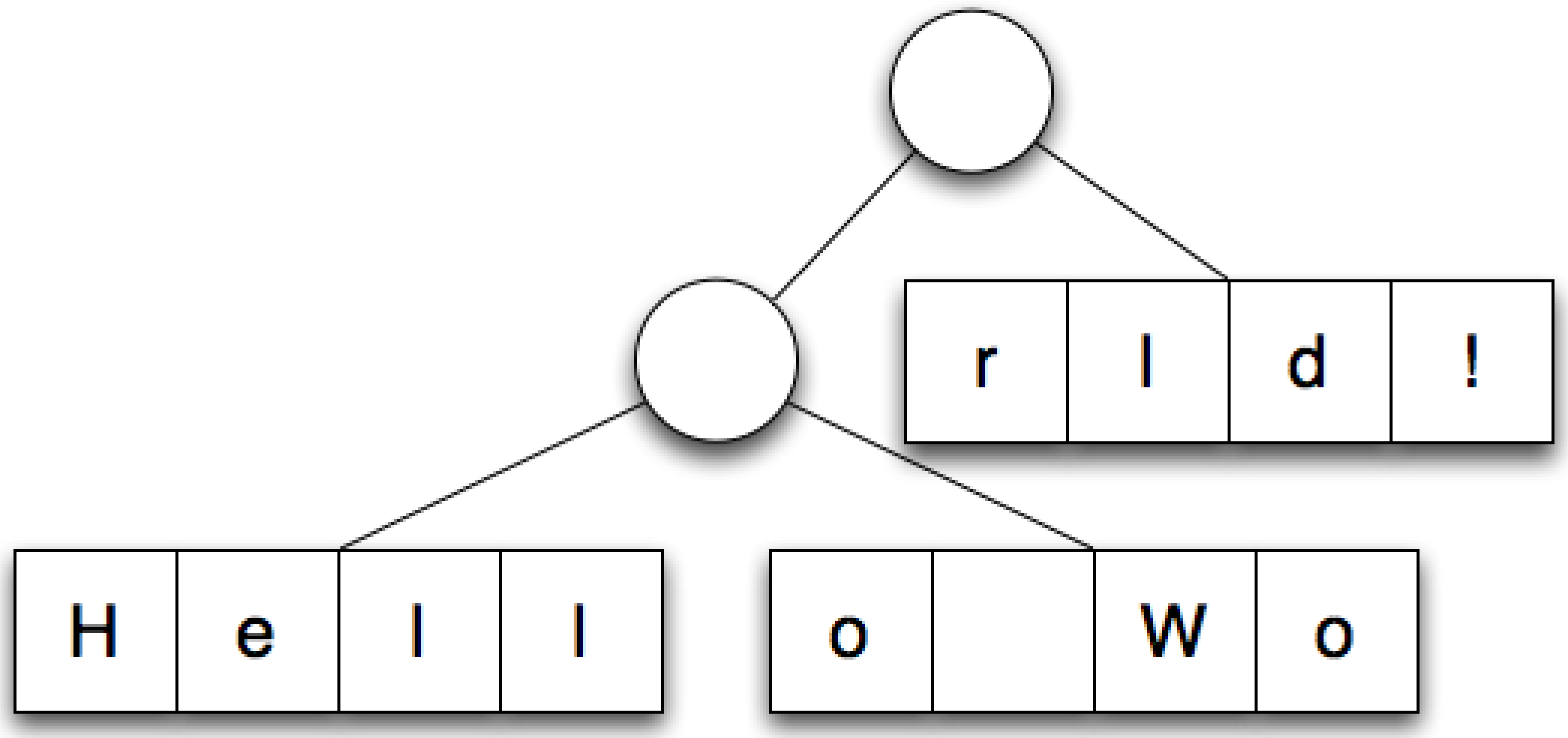


© 2016 Adobe Systems Incorporated. All Rights Reserved.

More Complex Word Model

- Need to be able to set the selection in “constant” time
  - This would imply a vector data structure
- Also need constant time insert and erase
  - This would imply a list data structure

- Solution: a more complex data structure such as a rope



© 2016 Adobe Systems Incorporated. All Rights Reserved.

"I don't smoke, I don't drink… I recycle…"

– 50/50

Good Code

Good code is *correct*

Consistent; without contradiction

Good code has *meaning*

Correspondence to an entity; specified, defined

Good code is *efficient*

Maximum effect with minimum resources

Good code is r*eusable*

Applicable to multiple problems; general in purpose

© 2016 Adobe Systems Incorporated. All Rights Reserved. Adobe Confidential.

Reusable

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Reusable

Concrete but of general use, i.e. numeric algorithms, utf conversions, …

Generic when algorithm is useful with different models

- Sometimes faster to convert one model to another

Runtime dispatched when types not known at compile time

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Reusable

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Reusable

Minimize client dependencies and intrusive requirements

Separate data structures from algorithms

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Reusable

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Reusable

```
template <class T, class InputIterator, class OutputIterator>
OutputIterator copy_utf(InputIterator first, InputIterator last,
        OutputIterator result);


const char str[] = u8"Hello World!";
vector<uint16_t> out;
copy_utf<uint16_t>(begin(str), end(str), back_inserter(out));
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

"You mean we're in the future."

– Back to the Future Part II

Why Status Quo Will Fail

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Why Status Quo Will Fail

"I've assigned this problem [binary search] in courses at Bell Labs and IBM. Professional programmers had a couple of hours to convert the description into a programming language of their choice; a high-level pseudo code was fine... Ninety percent of the programmers found bugs in their programs (and I wasn't always convinced of the correctness of the code in which no bugs were found)."

– Jon Bentley, Programming Pearls, 1986

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Why Status Quo Will Fail

```
int* lower_bound(int* first, int* last, int value)
{
    while (first != last)
    {
        int* middle = first + (last - first) / 2;

        if (*middle < value) first = middle + 1;
        else last = middle;
    }

    return first;
}
```

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Signs of Hope

Elements of Programming

Concepts aren't dead yet in C++

Increased interest in new languages and formalisms

Renewed interest in Communication Sequential Processes

Renewed interest in Functional Programming ideas

Rise of Reactive Programming & Functional Reactive Programming

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Work Continues

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Work Continues

Generating Reactive Programs for Graphical User Interfaces from Multi-way Dataflow Constraint Systems, GPCE 2015, Gabriel Foust, Jaakko Järvi, Sean Parent

One Way To Select Many, ECOOP 2016, Jaakko Järvi, Sean Parent

https://github.com/sean-parent/sean-parent.github.io/wiki/Papers-and-Presentations

https://github.com/stlab

© 2016 Adobe Systems Incorporated. All Rights Reserved.

Write Better Code

© 2016 Adobe Systems Incorporated. All Rights Reserved. Adobe Confidential.

Adobe