Flexibilité et Performance dans l'implémentation de JS

Sémantique de l'envoi de message et mémoization au site d'appel

Contexte

- Problématique
- Cause
- Approche proposée
- Définitions de Flexibilité et Performance
- Problème du jour

Contexte

- **Problématique**
- Cause
- Approche proposée
- Définitions de Flexibilité et Performance
- Problème du jour

Problématique

- Aucune étude à grande échelle du comportement de programmes JS depuis 2010 (Richards PLDI 2010)
- Absence de données sur leur évolution temporelle
- Comparaison inexistante du comportement des benchmarks améliorés avec le comportement identifié précédemment (Richards PLDI 2012 et JSMeter Ratanaworabhan 2010)

Problématique (2)

- Analyse réservée aux sites les plus populaires
- Analyse doit être réalisée par une équipe de chercheurs experts
- Infrastructure opaque et réservée à un usage unique

Contexte

- Problématique
- **Cause**
- Approche proposée
- Définitions de Flexibilité et Performance
- Problème du jour

Cause

- Approche utilisée requiert la modification d'une implémentation complexe et optimisée pour la performance
  - Difficulté à maintenir à jour
  - Requiert l'installation et l'utilisation d'une version maison d'un navigateur web
  - Difficulté à exposer en JS les fonctionnalités ajoutées

Contexte

- Problématique
- Cause
- **Approche proposée**
- Définitions de Flexibilité et Performance
- Problème du jour

Approche proposée

- Implémentation méta-circulaire de JS s'exécutant dans le navigateur web, exposant les fonctionnalités nécessaires à l'analyse de manière suffisamment performante pour pouvoir être utilisée sur des sites existants

Avantages

- Élimine la dépendance à l'implémentation de la VM
  - Facilite l'évolution
- Évite la réimplémentation de nombreuses fonctionnalités (Flottants, GC, Dates, RegExp, etc.)
- Permet la modification du comportement en JS
  - Facilite l'usage de l'infrastructure dans d'autres contextes

Avantages (2)

- Permet la réutilisation des optimisations de la VM (OSR, GC Gen., Inlining, Reg. Alloc., Op. Arithm.,etc.)
  - Profite de l'évolution en performance au lieu de compétitionner

Principes de design

- Unifier toutes les opérations du modèle objet de JS comme envoi de message (appel de méthode)
  - Exposer les opérations comme des méthodes sur les objets
- Unifier tous les appels de fonction comme l'envoi du message "call" à l'objet fonction concerné
- Utiliser les objets primitifs (nombres, chaînes de caractères, booléens, etc.) et les primitives de flot de contrôle tel quel

Contexte

- Problématique
- Cause
- Approche proposée
- **Définitions de Flexibilité et Performance**
- Problème du jour

Déf. Flexibilité

- Ouverture (*Openness*)
  - Les éléments d'intérêt sont accessibles et modifiables en JS
- Dynamisme
  - Leur comportement peut être modifié pendant l'exécution du programme
    - Granularité temporelle
- Extensibilité
  - Les éléments (nouveaux ou existants) du système sont définis de manière différentielle à partir d'autres éléments
    - Granularité spatiale

Déf. Performance

- Coût de la flexibilité inutilisée devrait être minimal en temps et en espace
- L'expérience utilisateur de navigation devrait être minimalement perturbée
  - délai de réponse devrait être court
  - la performance en régime permanent devrait être excellente avant ou après instrumentation
  - l'implémentation devrait permettre de minimiser le coût d'instrumentation
- Une application web devrait être en mesure de s'exécuter à l'intérieur de l'espace mémoire disponible

Contexte

- Problématique
- Cause
- Approche proposée
- Définitions de Flexibilité et Performance
- **Problème du jour**

Problème du jour

Marier flexibilité et performance dans l'envoi de messages

- Flexibilité:
  - Permettre l'instrumentation dynamique du protocole d'appel ("call")
  - Permettre la mémoization des opérations de base
- Performance:
  - Minimiser le coût des envois lorsqu'un ou plusieurs des éléments suivants sont constants: le nom du message, la valeur de la méthode, le receveur
  - Minimiser le coût de l'instrumentation du protocole d'appel lorsqu'il n'est pas sensible au contexte

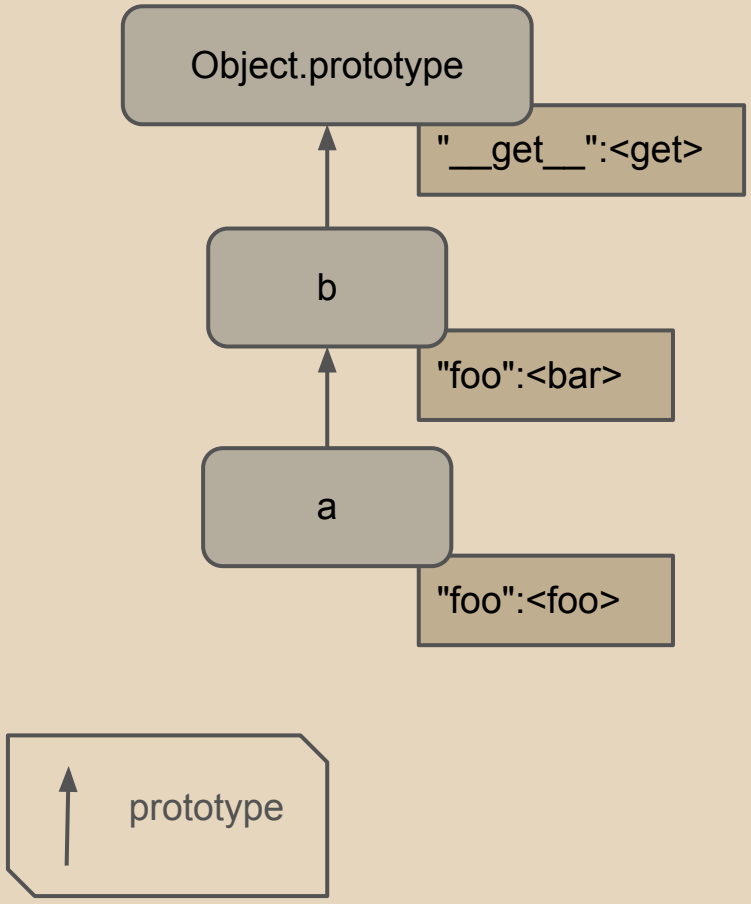
Plan de la présentation

- Envoi de message en JS
- Augmenter la flexibilité
- Exploiter la stabilité
- Implémentation
- Évaluation
- Limitations et Pistes

Plan de la présentation

- **Envoi de message en JS**
- Augmenter la flexibilité
- Exploiter la stabilité
- Implémentation
- Évaluation
- Limitations et Pistes

Envoi de message en JS

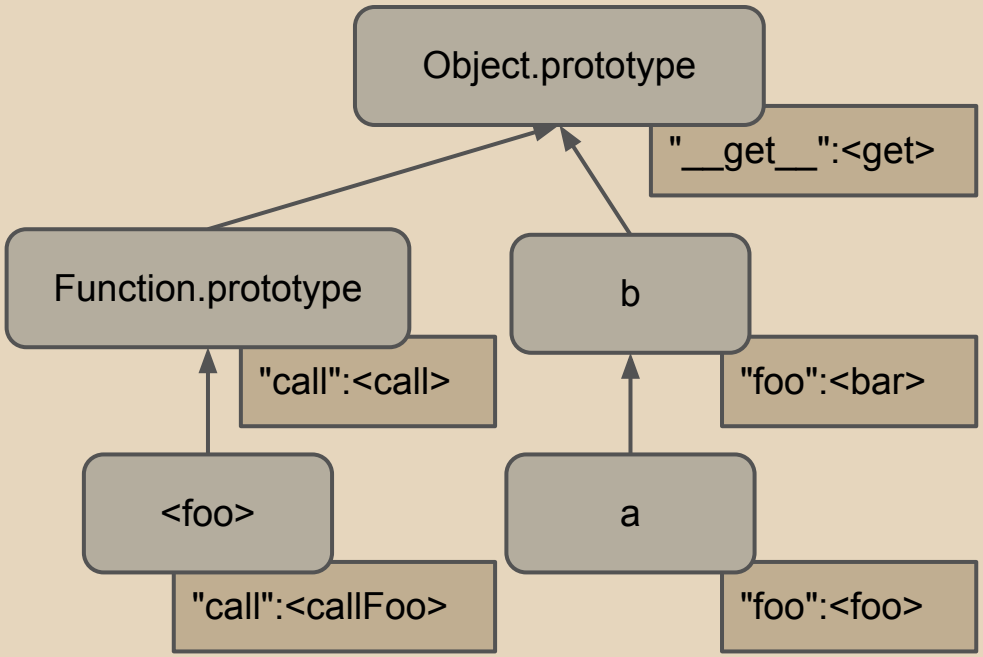



```
function bind(rcv, msg) {
    var obj = rcv;
    while (obj !== null) {
        if (obj.has(msg))
            return obj.get(msg);
        obj = obj.prototype;
    };
    return null;
}
function send(rcv, msg, ..args) {
    var m = bind(rcv, msg);
    return m.call(
        rcv,
        ..args
    );
}
```

Plan de la présentation

- Envoi de message en JS
- **Augmenter la flexibilité**
- Exploiter la stabilité
- Implémentation
- Évaluation
- Limitations et Pistes

Augmenter la flexibilité



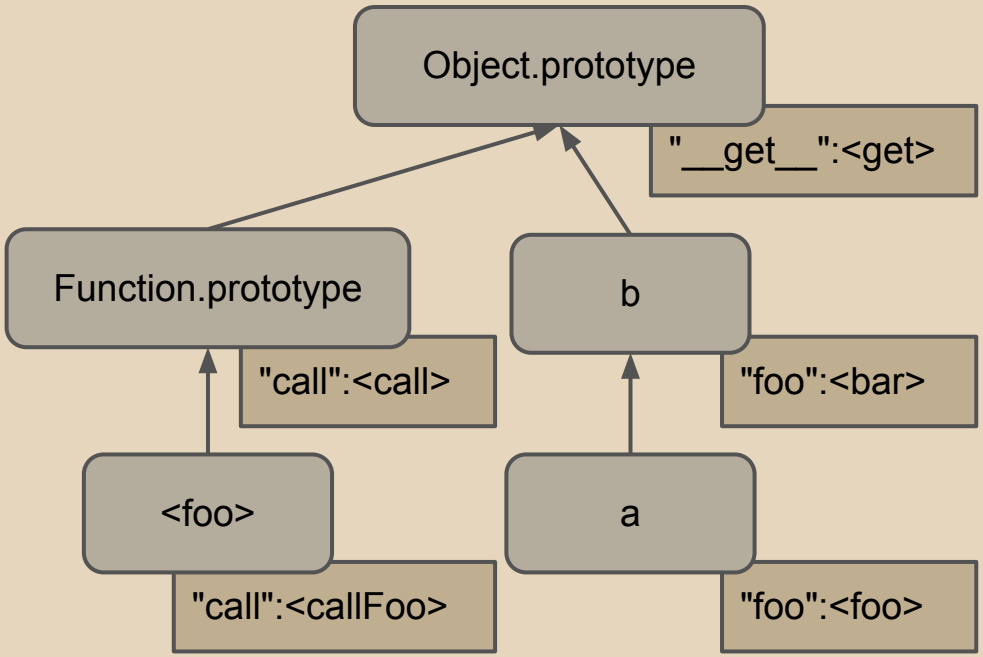

```
function bind(rcv, msg) {...}
function <call>(rcv, ..args) {
    return this.call(rcv, ..args);
}
function send(rcv, msg, ..args) {
    var m      = bind(rcv, msg);
    var callFn = bind(m, "call");
    return callFn.call(
        m,
        rcv,
        ..args
    );
}
```

Ex: Graphe d'appels dynamiques

```
function callInstr(rcv, ..args) {
    try {
        beforeCall(this);
        var result =
            this.call(rcv, ..
        args);
     } finally {
        afterCall(this);
    }
    return result;
}
```

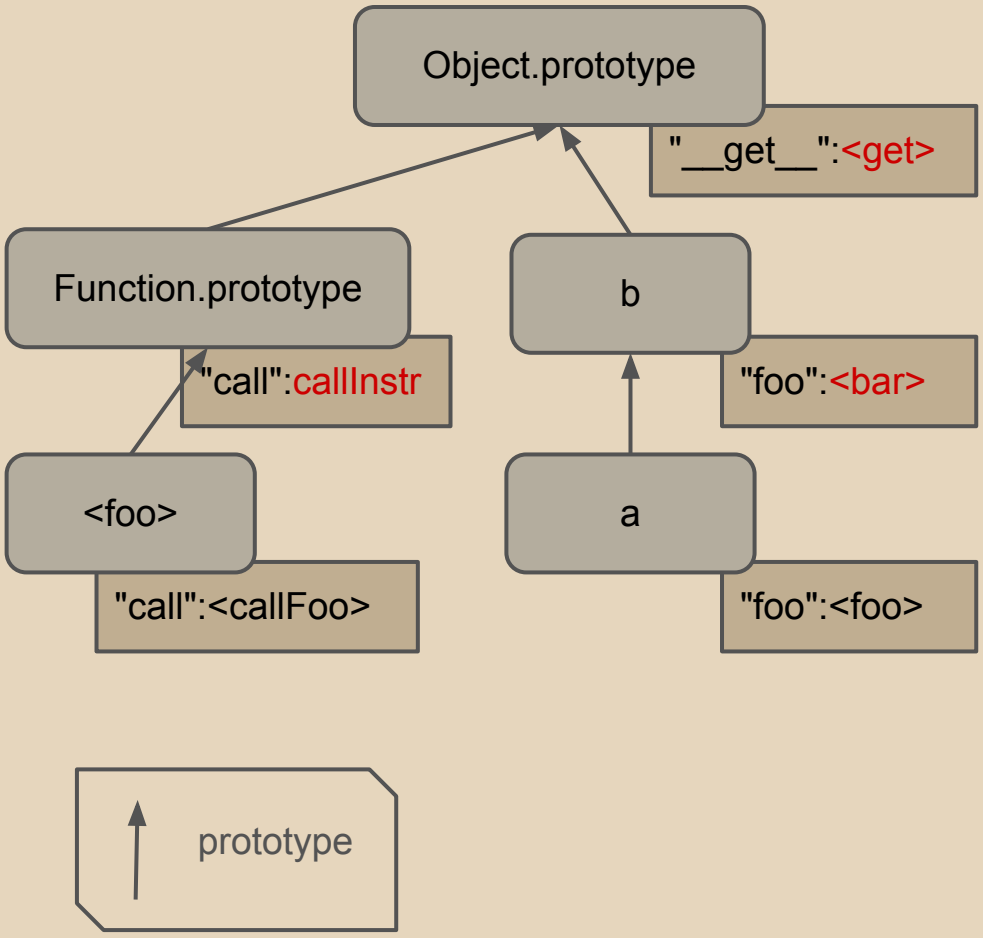
```
var callStack = ["global"];
var callGraph = new Graph();

function beforeCall(fn) {
    callGraph
    .node(callStack.top())
    .addEdgeTo(fn.__id__);
    callStack.push(fn);
}
function afterCall(fn) {
    callStack.pop();
}
```

Augmenter la flexibilité



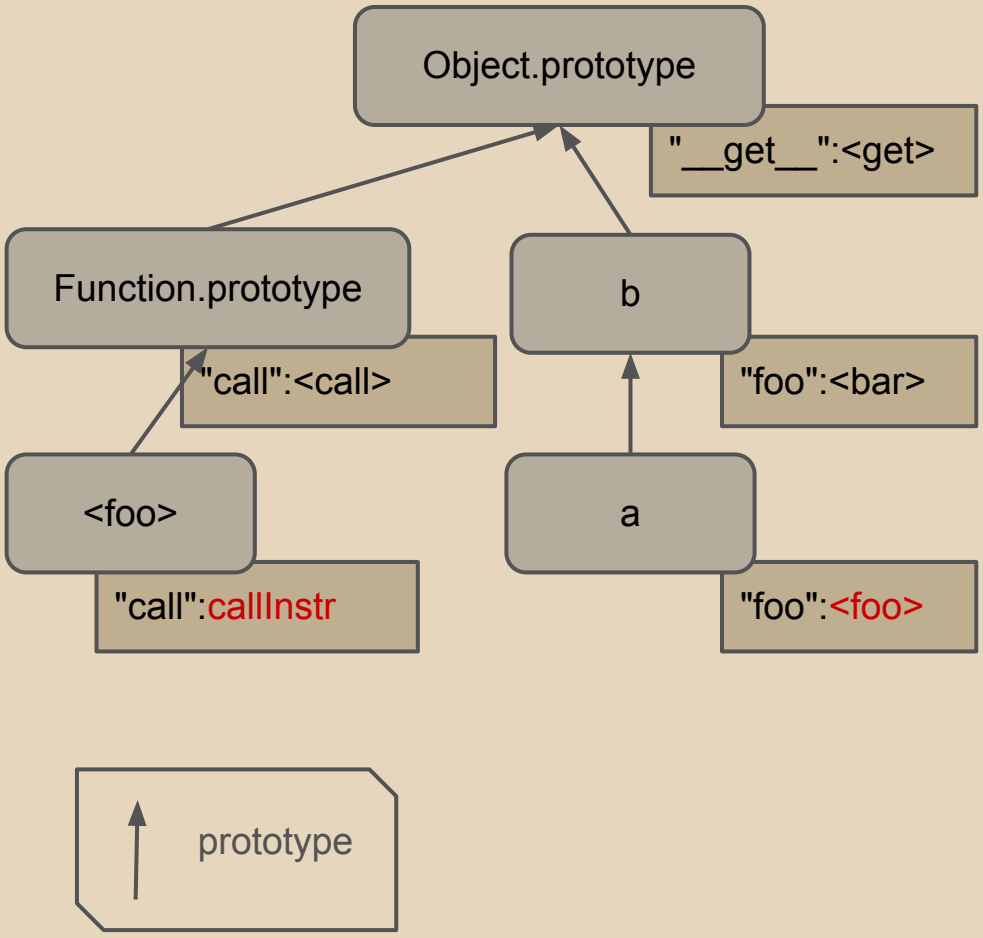

```
function bind(rcv, msg) {...}
function <call>(rcv, ..args) {
    return this.call(rcv, ..args);
}
function send(rcv, msg, ..args) {
    var m      = bind(rcv, msg);
    var callFn = bind(m, "call");
    return callFn.call(
        m,
        rcv,
        ..args
    );
}
```

Augmenter la flexibilité




```
function bind(rcv, msg) {...}
function <call>(rcv, ..args) {
    return this.call(rcv, ..args);
}
function send(rcv, msg, ..args) {
    var m      = bind(rcv, msg);
    var callFn = bind(m, "call");
    return callFn.call(
        m,
        rcv,
        ..args
    );
}
```

Augmenter la flexibilité

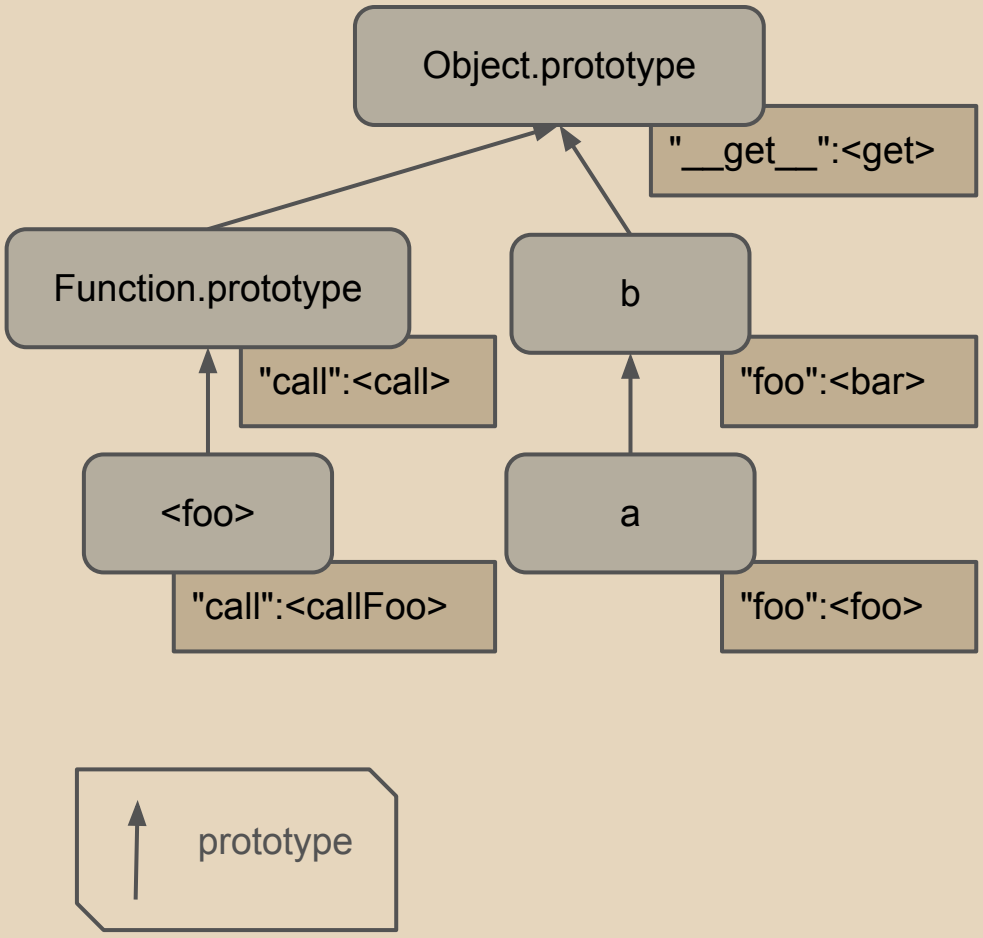



```
function bind(rcv, msg) {...}
function <call>(rcv, ..args) {
    return this.call(rcv, ..args);
}
function send(rcv, msg, ..args) {
    var m      = bind(rcv, msg);
    var callFn = bind(m, "call");
    return callFn.call(
        m,
        rcv,
        ..args
    );
}
```

Plan de la présentation

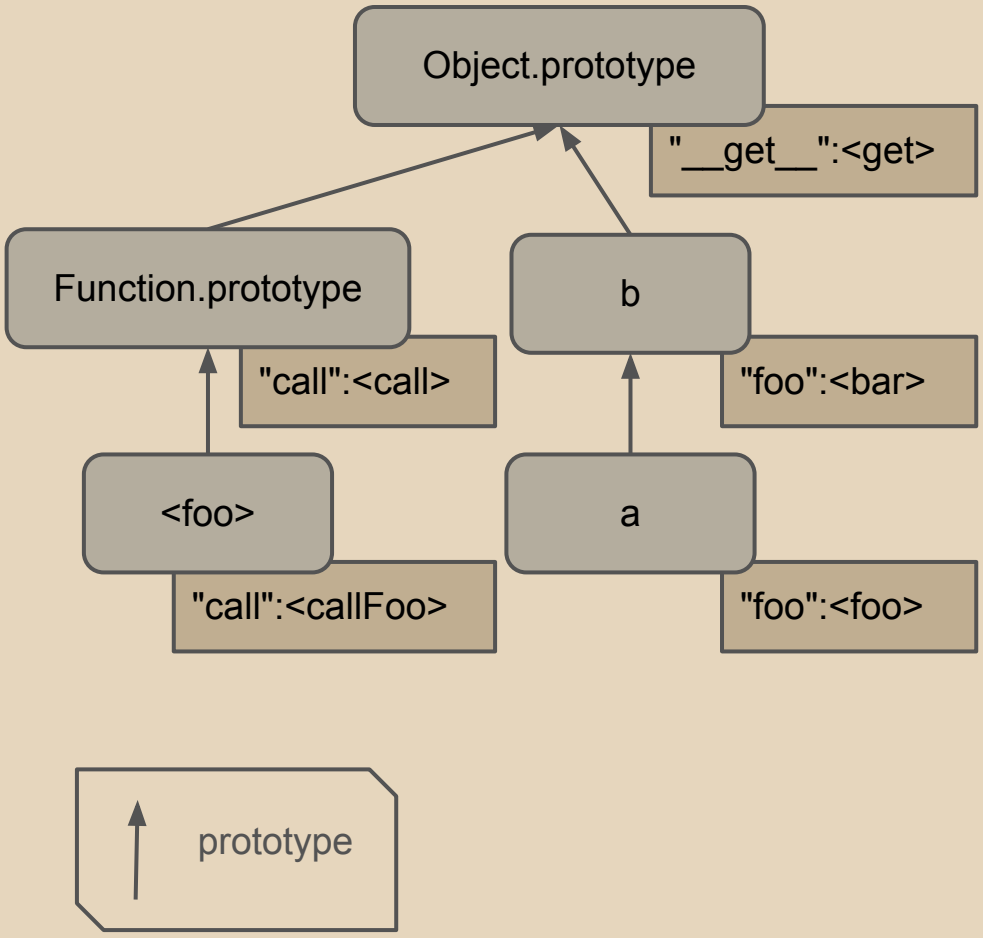
- Envoi de message en JS
- Augmenter la flexibilité
- **Exploiter la stabilité**
- Implémentation
- Évaluation
- Limitations et Pistes

Insérer l'appel directement




```
function bind(rcv, msg) {...}
function <call>(rcv, ..args) {...}
function send(rcv, msg, ..args) {
    var m      = bind(rcv, msg);
    var callFn = bind(m, "call");
    if (callFn === <call>) {
        return m.call(rcv, ..args);
    } else {
        return callFn.call(
            m,
            rcv,
            ..args
        );
    }
}
```

Mémoizer la recherche




```
function bind(rcv, msg) {...}
function <call>(rcv, ..args) {...}
function send(rcv, msg, ..args) {
    var m      = bind(rcv, msg);
    var callFn = bind(m, "call");
    if (callFn === <call>) {
        return m.call(rcv, ..args);
    } else {
        return callFn.call(
            m,
            rcv,
            ..args
        );
    }
}
```

Mémoizer la méthode aussi!

```
function instrCall(rcv, ..args) {
    try {
        beforeCall(this);
        var result =
            this.call(rcv, ..
        args);
     } finally {
        afterCall(this);
    }
    return result;
}
```

```
var callStack = ["global"];
var callGraph = new Graph();

function beforeCall(fn) {
    callGraph
    .node(callStack.top())
    .addEdgeTo(fn.__id__);
    callStack.push(fn);
}
function afterCall(fn) {
    callStack.pop();
}
```

Mémoizer la méthode aussi! (2)

```
function <get>(name) {
    var obj = this;
    if (obj.has(name))
        return obj.get(name);
    obj = obj.prototype;
    while (obj !== null) {
        if (obj.has(name))
            return obj.get(name);
        obj = obj.prototype;
    }
    return undefined;
}
```

Mémoizer l'exécution de la méthode aussi! (3)

```
function <get>(name) {...}
set(Function.prototype, "__memoize__",
  function () { return this; }
);
set(<get>, "__memoize__",
  function (rcv, args) {
    var memName = args[0];
    if (rcv.has(memName))
      return function (name) {
        return (name === memName)?
            this.get(name) :
            <get>.call(this, name);
      };
    else
      return this;
  }
);
```

```
function send(rcv, msg, ..args) {
     var m      = bind(rcv, msg);
     var callFn = bind(m, "call");
     var memMeth = send(
          m,"__memoize__",rcv,args);
     var memCall = send(
          callFn, "__memoize__", m,
          [m].concat(args));
     if (memCallFn === <call>) {
          // Mémoizer memMeth au SA
          return m.call(rcv, ..args);
     } else {
          // Mémoizer memCall à m au SA
          return callFn
               .call(m,rcv,..args);
     }
}
```

Plan de la présentation

- Envoi de message en JS
- Augmenter la flexibilité
- Exploiter la stabilité
- **Implémentation**
- Évaluation
- Limitations et Pistes

Implémentation

- Code du site d'appel
  - Gestion des valeurs primitives
  - Gestion des messages non-existants
  - Gestion du receveur
  - Suivi des valeurs mémoizées
- Mécanisme de suivi et d'invalidation (sera traité dans une prochaine présentation)
  - Ajout de liens entre un site d'appel et les valeurs de propriété
  - Retrait de liens
  - Invalidation des caches lorsqu'une valeur suivie change ou qu'un objet change de structure et modifie les valeurs retournées par une recherche

Code du site d'appel

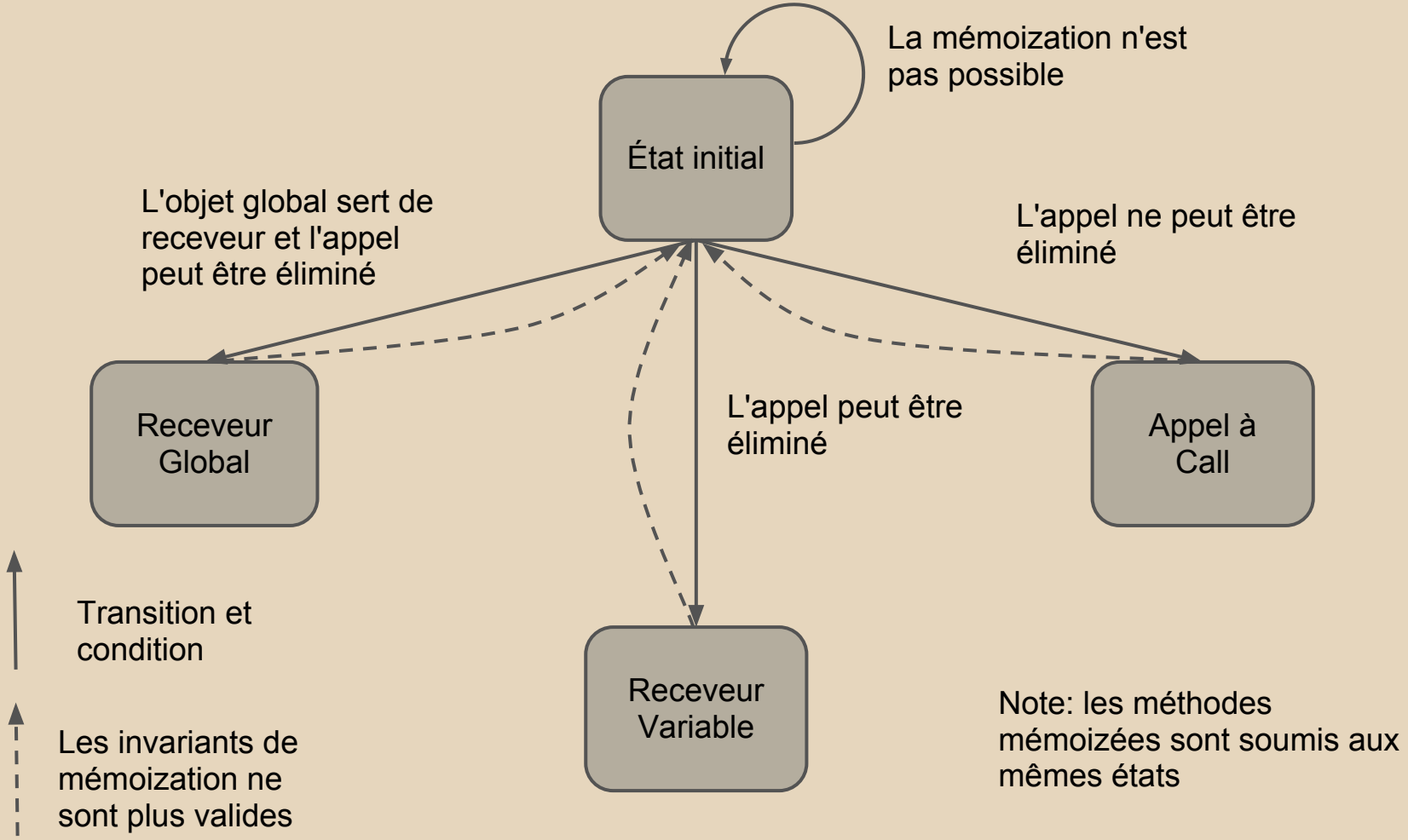
- Appel à une fonction globale respectant le protocole conventionnel (rcv, closure, .. args)
- L'argument *closure* est utilisé pour passer un tableau d'arguments propre au site d'appel, extensible
- Le site d'appel change d'état en modifiant la fonction globale correspondante

Code du site d'appel

```
function foo(x) { return x; }

print(foo(10));
```

```
var codeCache0 = initState;
var dataCache0 = [0, "foo"];
var codeCache1 = initState;
var dataCache1 = [1, "print"];

...

codeCache1(
    root_global,
    dataCache1,
    codeCache0(
        root_global,
        dataCache0,
        10
    )
);
```

États du site d'appel



Plan de la présentation

- Envoi de message en JS
- Augmenter la flexibilité
- Exploiter la stabilité
- Implémentation
- **Évaluation**
- Limitations et Pistes

Évaluation

Utilisation d'un benchmark qui fait l'exercice du protocole d'appel: *fibonacci de 40*

Version	Temps d'exécution (ms)	Vitesse relative à V8 (t/V8)
V8	1587	1
Photon Receveur Global	1872	1.18
Photon Receveur Variable	6619	4.17
Photon Call Redéfini*	9000	5.67
Photon Fib Mémoizant	48	0.03

Plan de la présentation

- Envoi de message en JS
- Augmenter la flexibilité
- Exploiter la stabilité
- Implémentation
- Évaluation
- **Limitations et Pistes**

Limitations et Pistes

- L'objet global comme receveur est supposé constant au site d'appel
- Les valeurs d'arguments constantes au site d'appel ne sont pas exploitées (Ex: obj.__get__("foo"))

- Piste de solution: Ajouter dans les données du site d'appel, le type des arguments (primitif constant, objet global, map de l'objet ou variable)

Au prochain épisode:

Le suivi et l'invalidation ou comment exploiter la stabilité temporaire dans un monde en mutation constante...