Identifying Non-Essential Changes Using Semantic Code Clone Detection

Yungbum Jung
Programming Language Laboratory
Seoul National University
@ROSAEC Workshop

의미 코드 쌍 탐지 기술로 불필요한 코드 변화 찾기

정영범

서울대학교 프로그래밍 연구실

6회 ROSAEC 워크샵 @ 파주

의미 코드 쌍 탐지 기술로 불필요한 코드 변화 찾기

박사 정영범

서울대학교 프로그래밍 연구실

6회 ROSAEC 워크샵 @ 파주

ICSE 2011 Hawaii

33rd International Conference on Software Engineering

Waikiki, Honolulu, Hawaii

May 21-28, 2011

MeCC: Memory Comparison-based Clone Detector*

Heejung Kim†, Yungbum Jung†, Sunghun Kim§, Kwangkeun Yi†
†Seoul National University, Seoul, Korea
{hjkim,dreameye,kwang}@ropas.snu.ac.kr
§The Hong Kong University of Science and Technology, Hong Kong
hunkim@cse.ust.hk

Solution

서울대 홍콩과기대

Non-Essential Changes in Version Histories

David Kawrykow and Martin P. Robillard
McGill University
Montréal, Canada
{dkawry,martin}@cs.mcgill.ca

Problem

맥길대


소프트웨어 패키지에서 “지식”을 수확해왔다.


소프트웨어 패키지에서 “지식”을 수확해왔다.
코드가 썩었나?
[Eick et al. 2001]

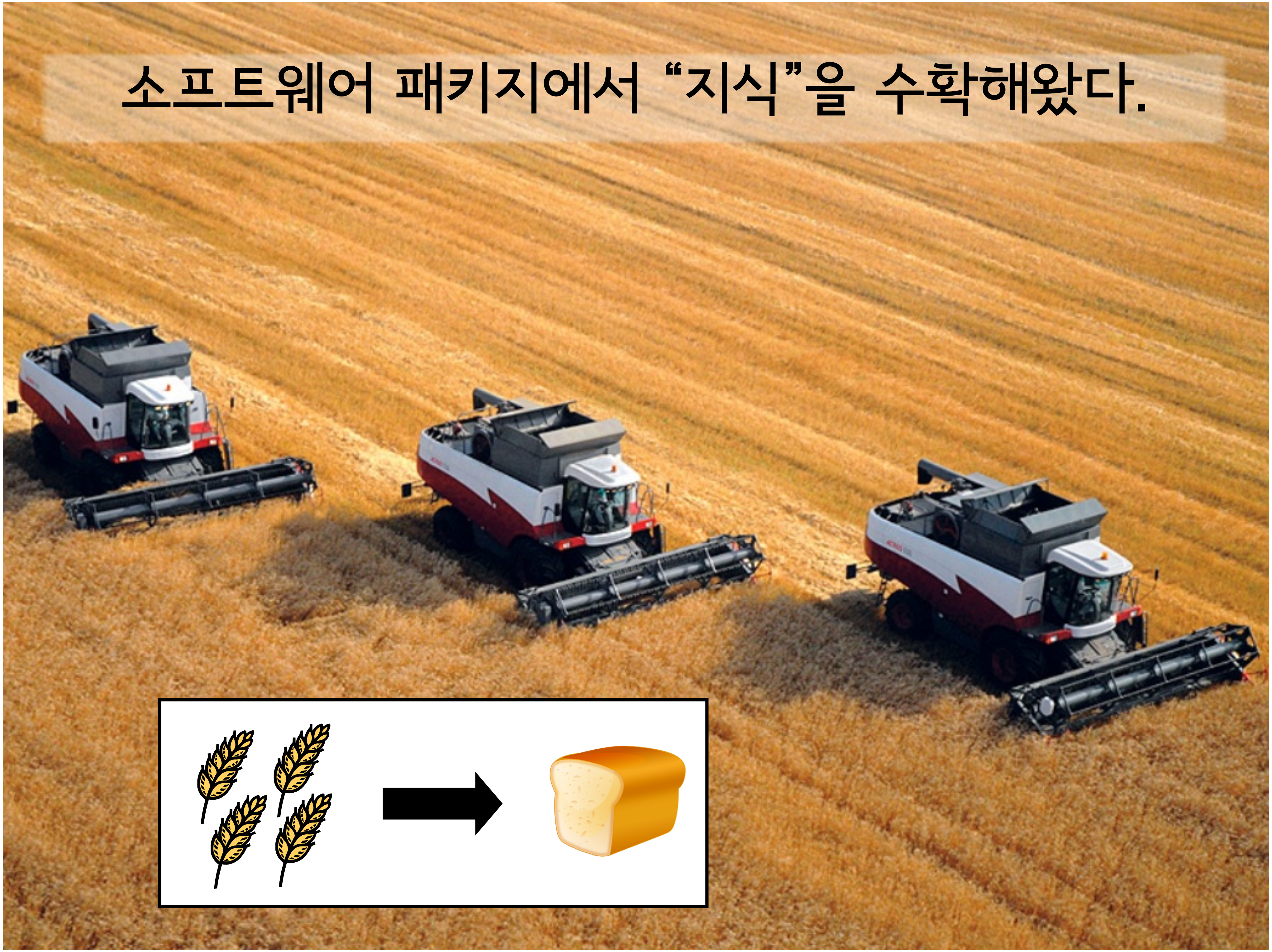
소프트웨어 패키지에서 “지식”을 수확해왔다.

함께 변경해야할 함수들 알려주기
[Zimmermann et al. 2004]


소프트웨어 패키지에서 “지식”을 수확해왔다.
버그 예측하기 등등...

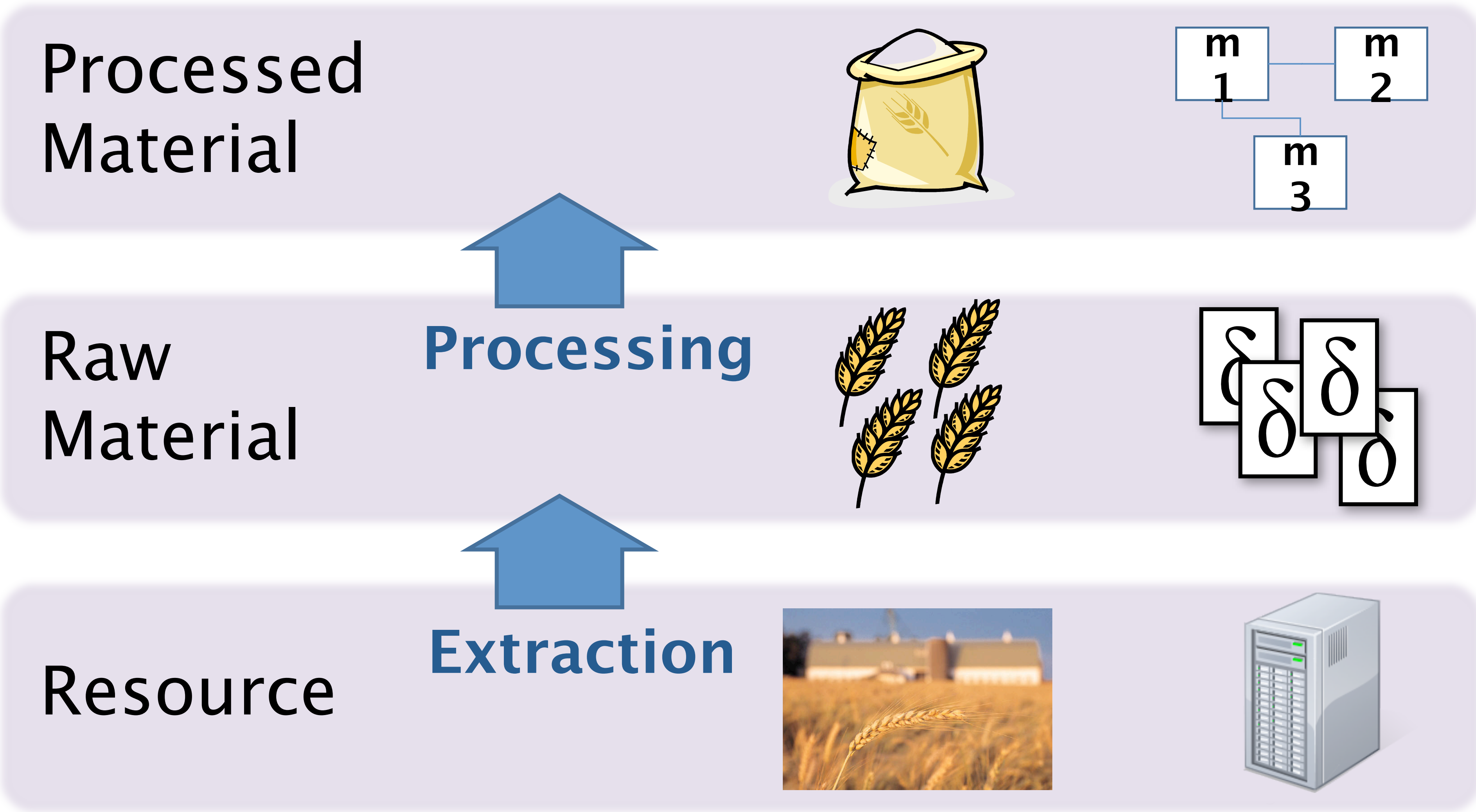

소프트웨어 패키지에서 “지식”을 수확해왔다.

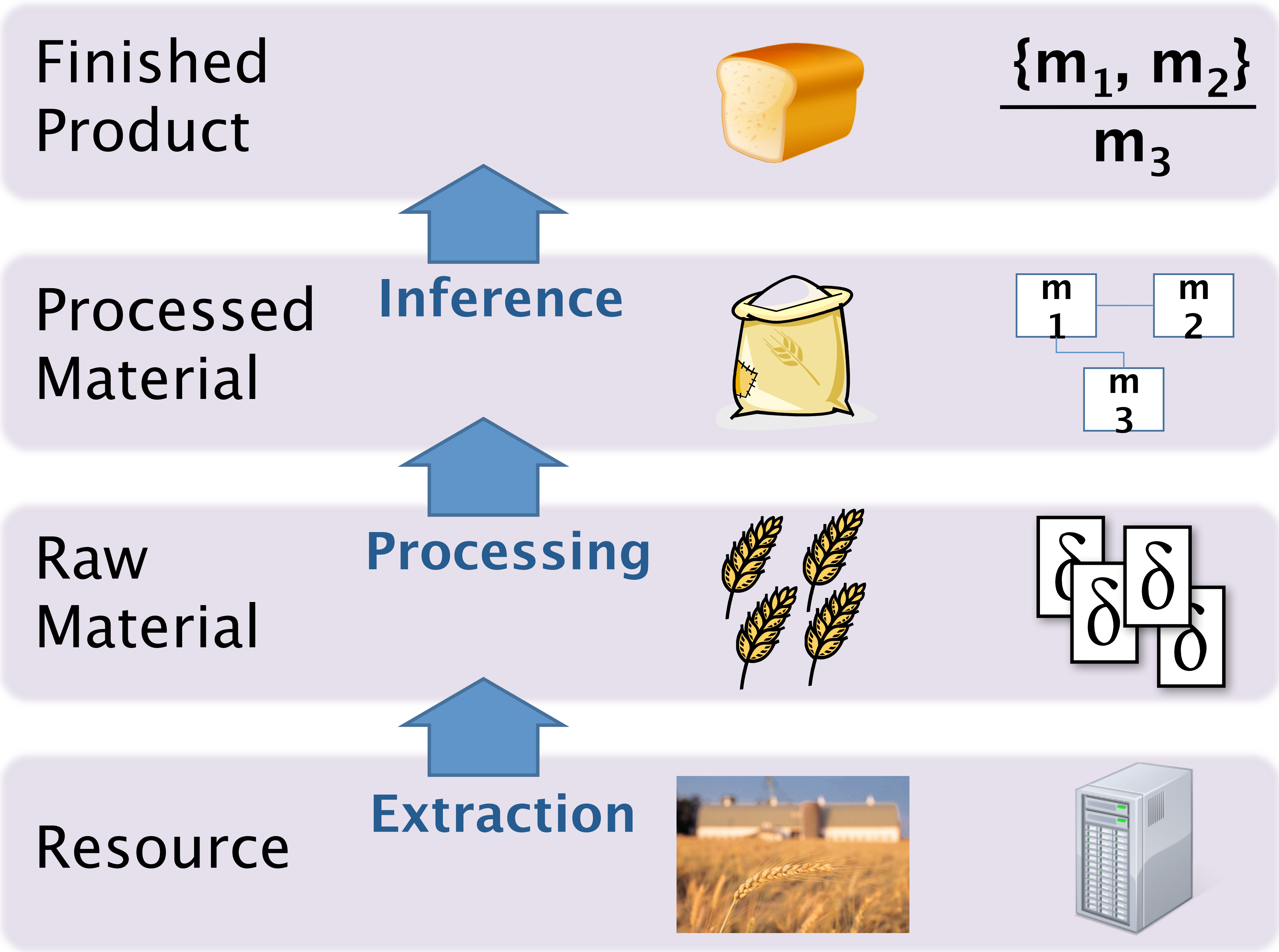
소프트웨어 패키지에서 “지식”을 수확해왔다.

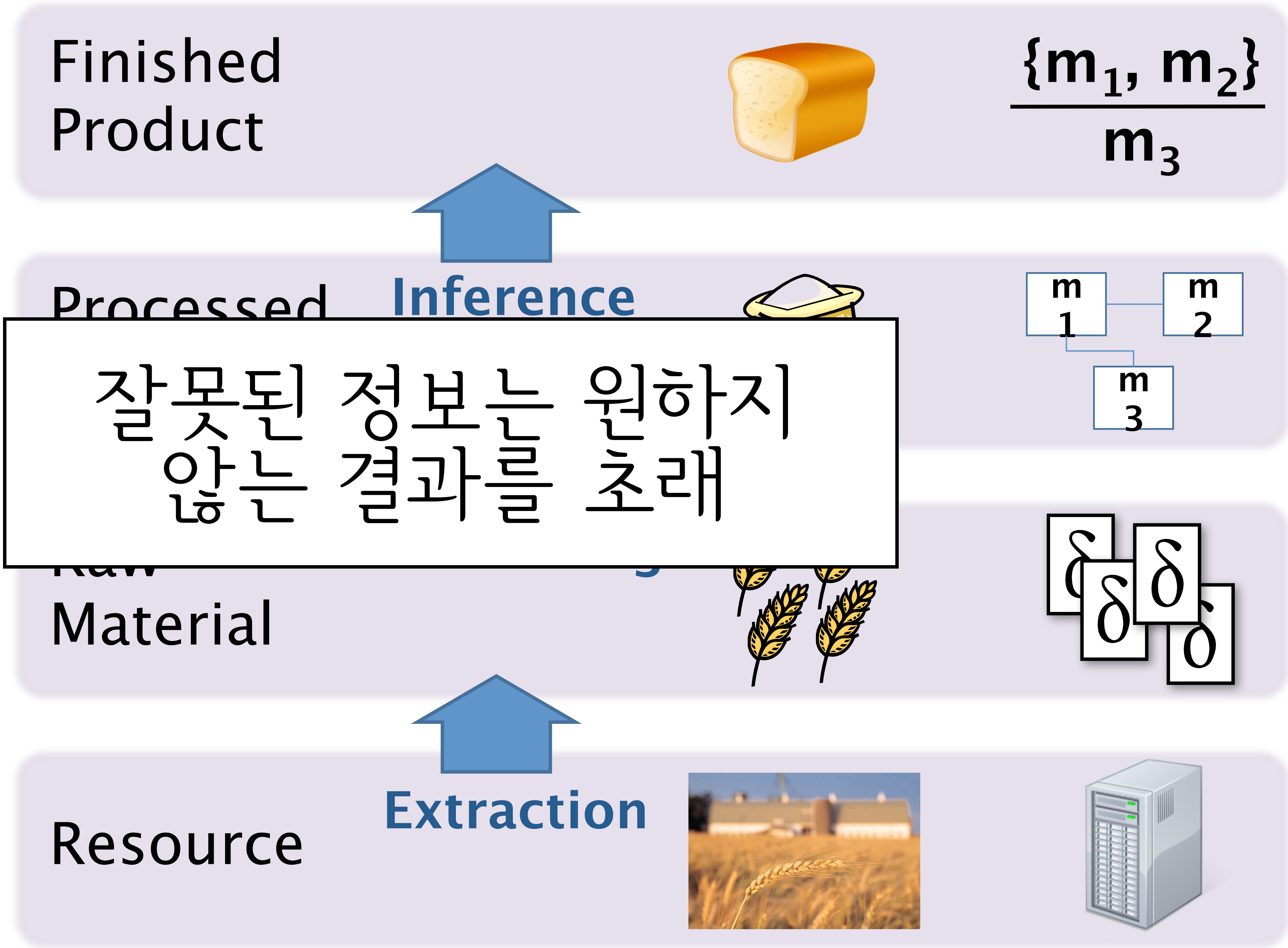


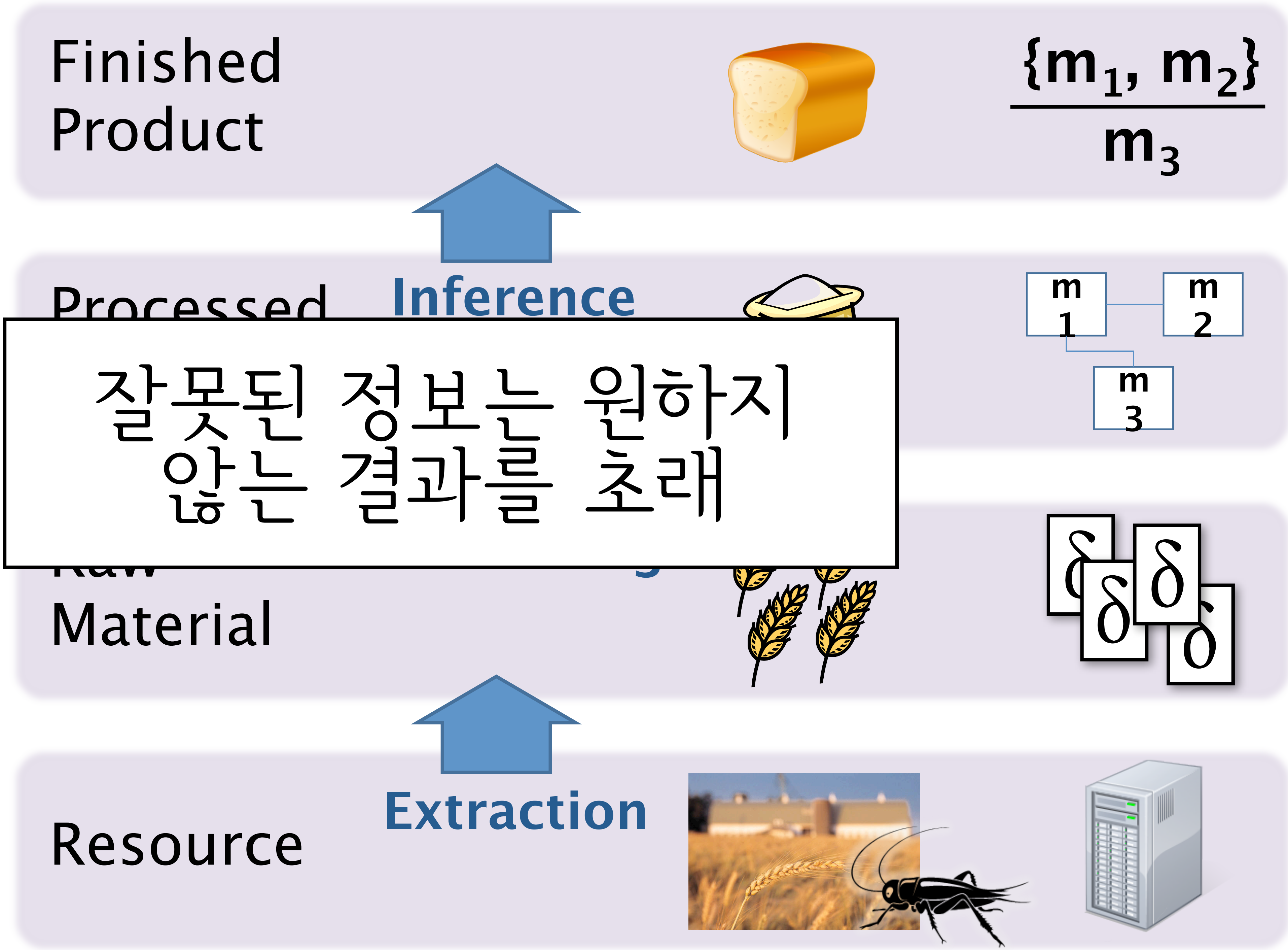
Resource

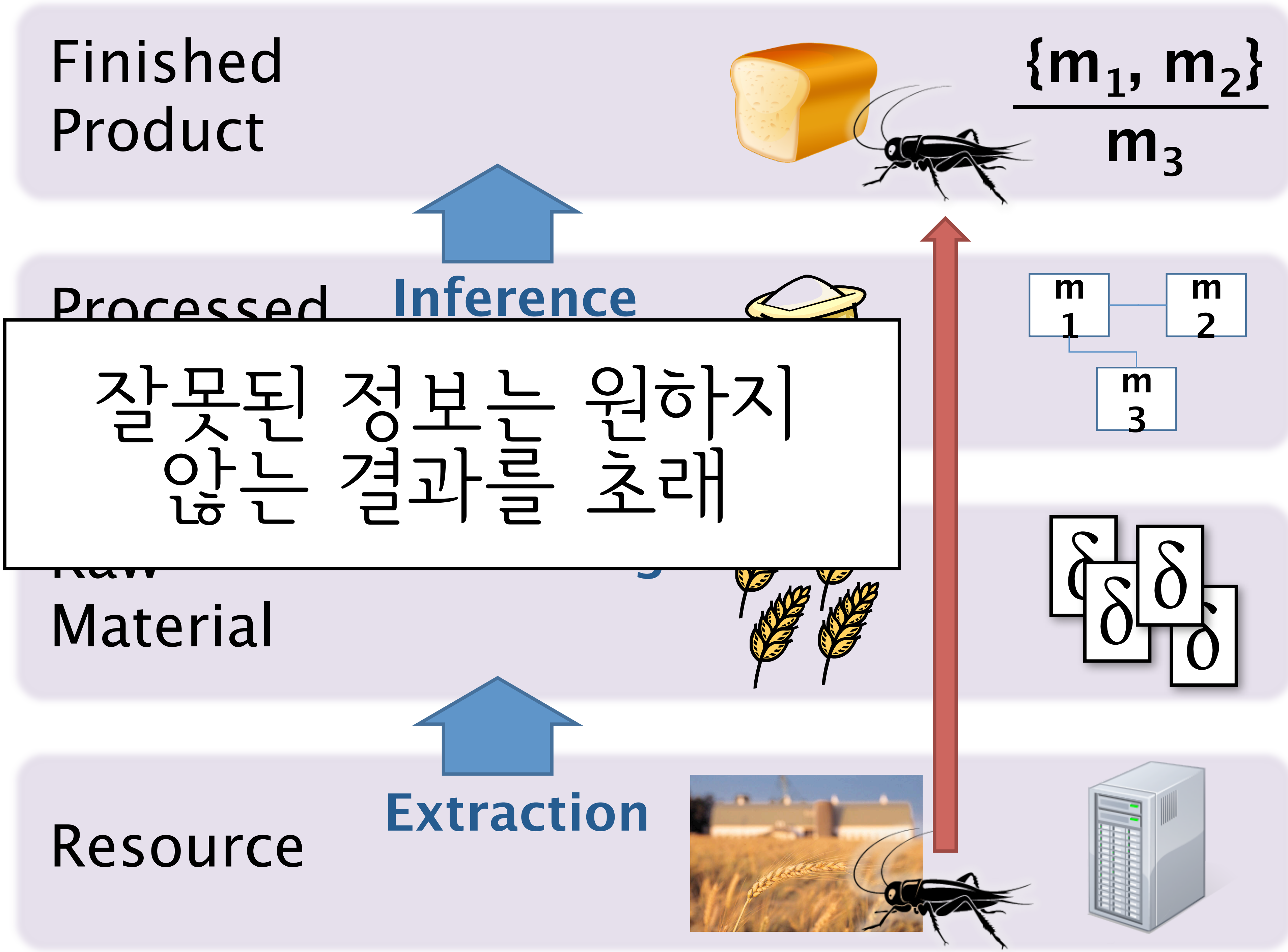

Raw Material
Extraction
Resource


Processed Material
Raw Material
Resource
Processing
Extraction
m 1
m 2
m 3
δ

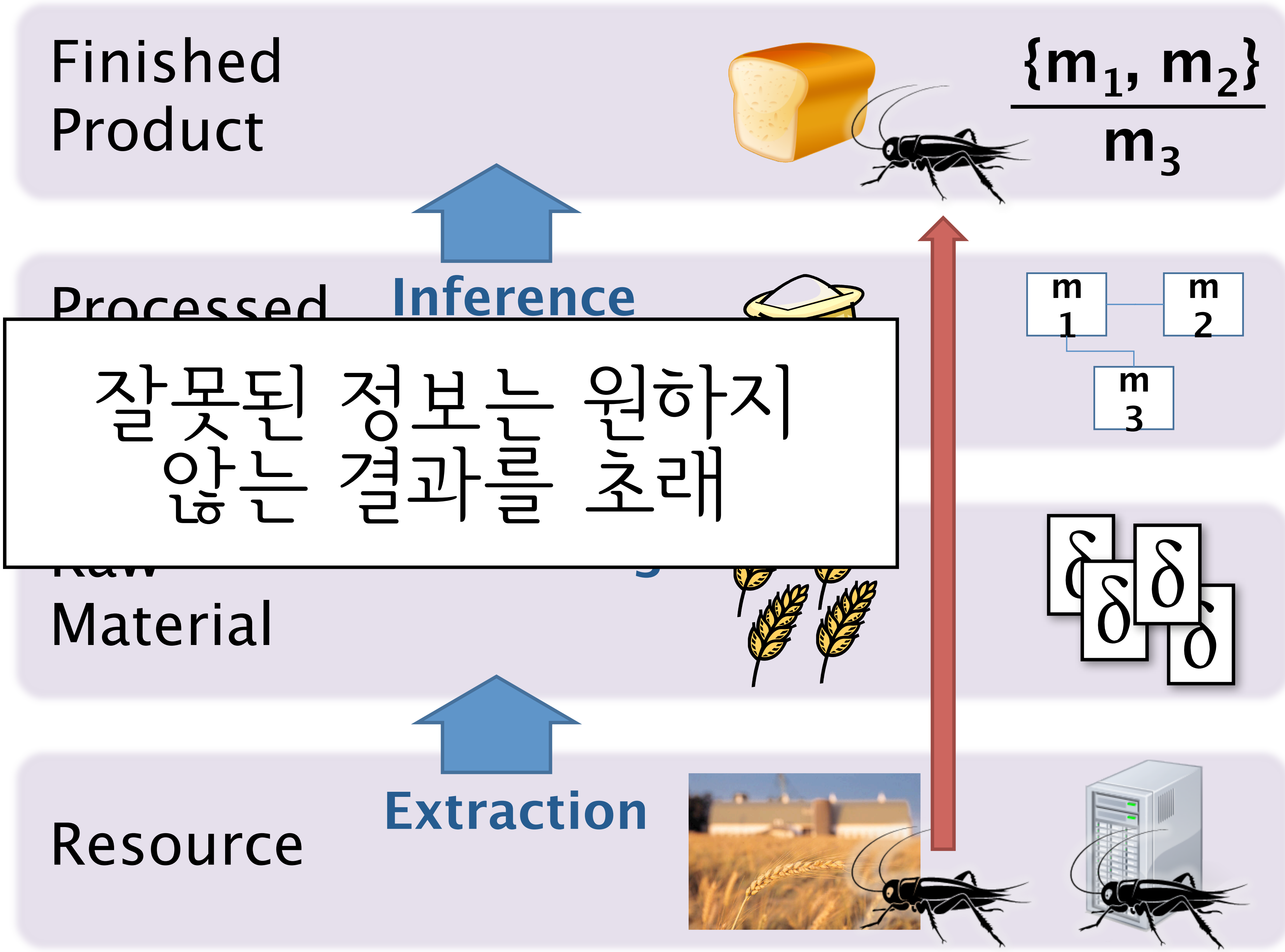

Finished Product
Processed Material
Raw Material
Resource
Inference
Processing
Extraction
$\frac{\{m_1, m_2\}}{m_3}$
m 1
m 2
m 3
δ δ δ δ

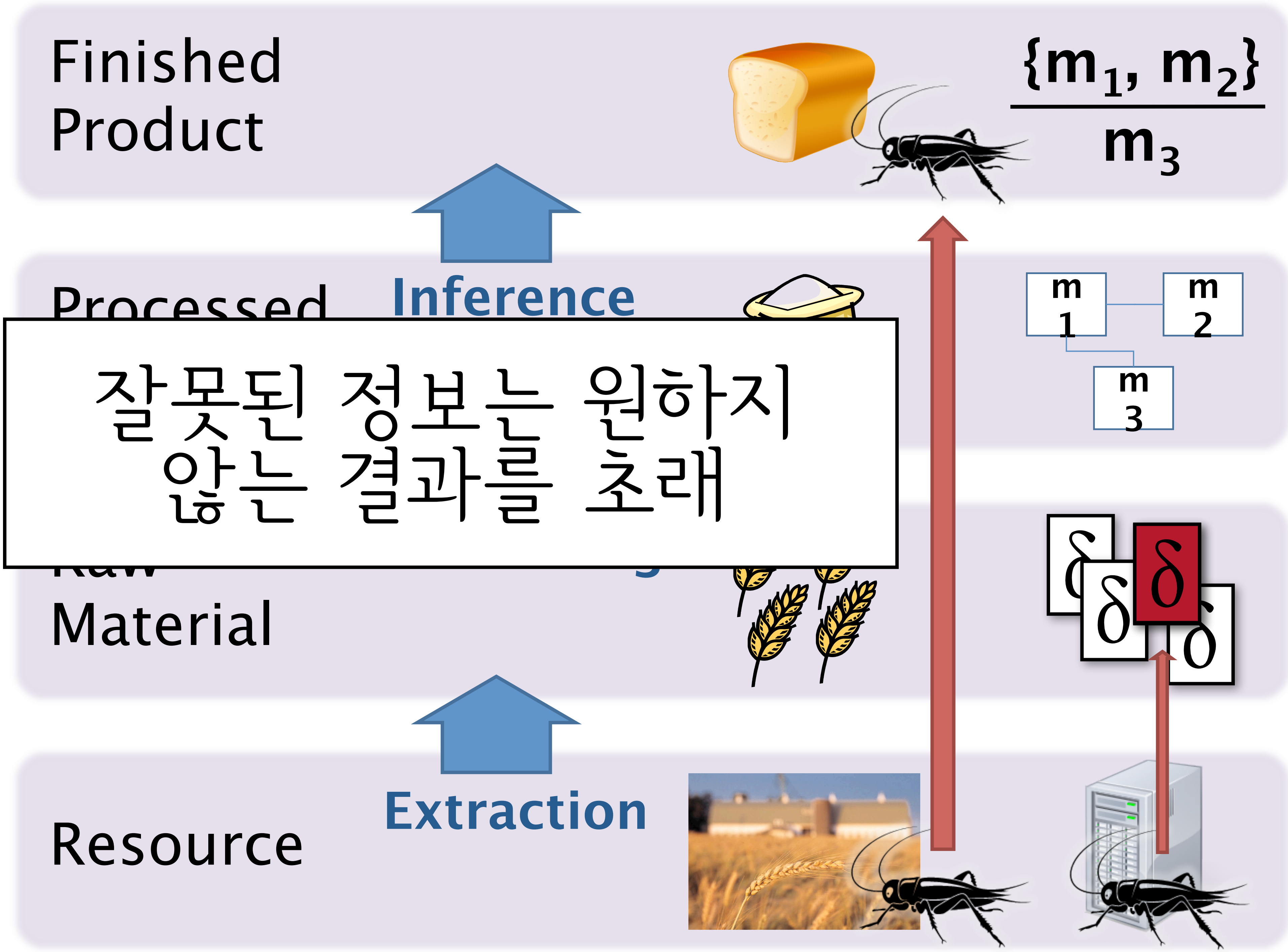

Finished
Product
{m1, m2} / m3
Processed
Inference
m 1
m 2
m 3
잘못된 정보는 원하지
않는 결과를 초래
Material
Extraction
Resource

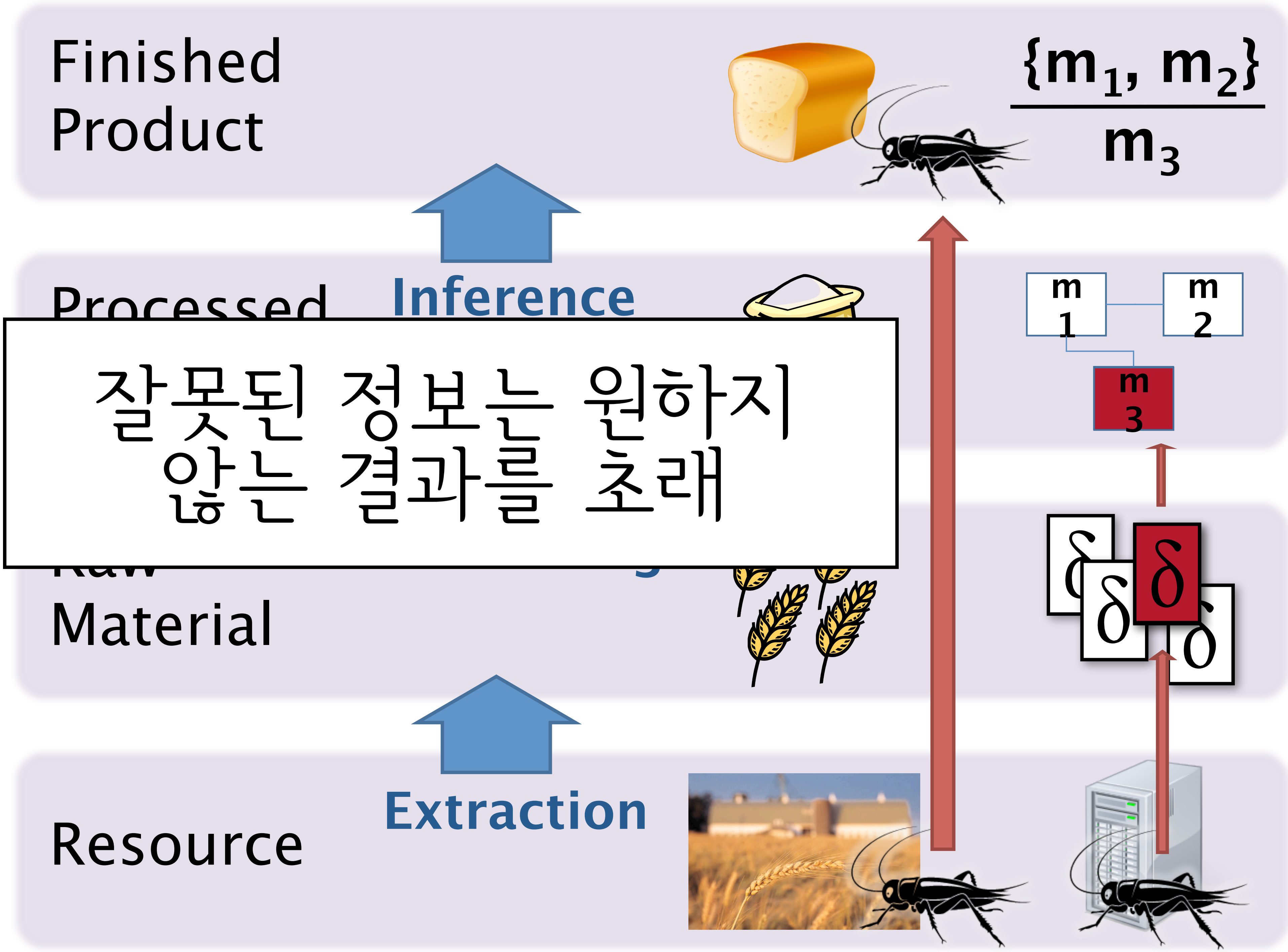

Finished Product
{m1, m2}
m3
Inference
Processed
m 1
m 2
m 3
잘못된 정보는 원하지 않는 결과를 초래
Material
δ
δ
δ
δ
Extraction
Resource

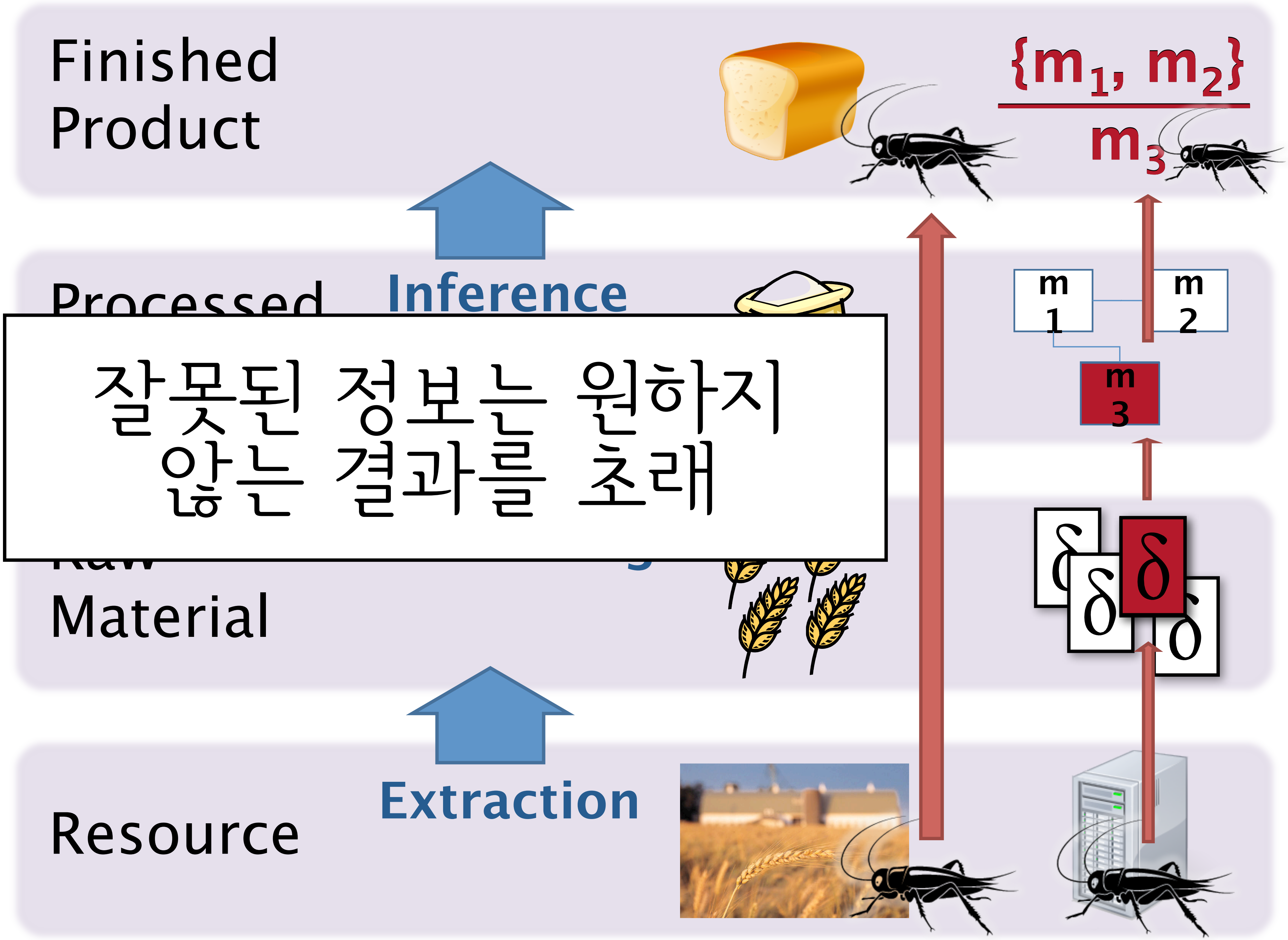


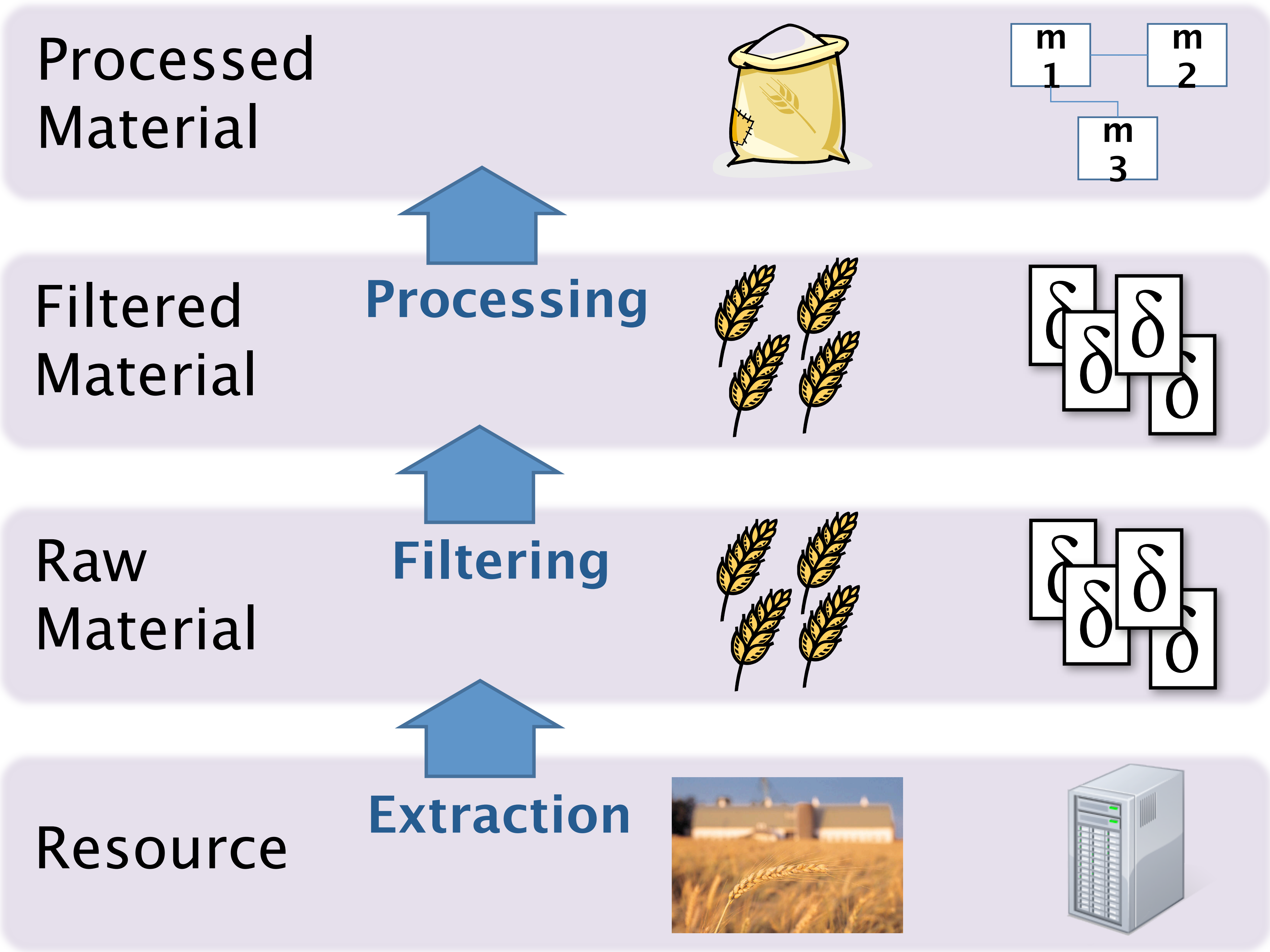
Finished Product
$\frac{\{m_1, m_2\}}{m_3}$
Processed
Inference
m 1
m 2
m 3
잘못된 정보는 원하지
않는 결과를 초래
Material
Extraction
Resource

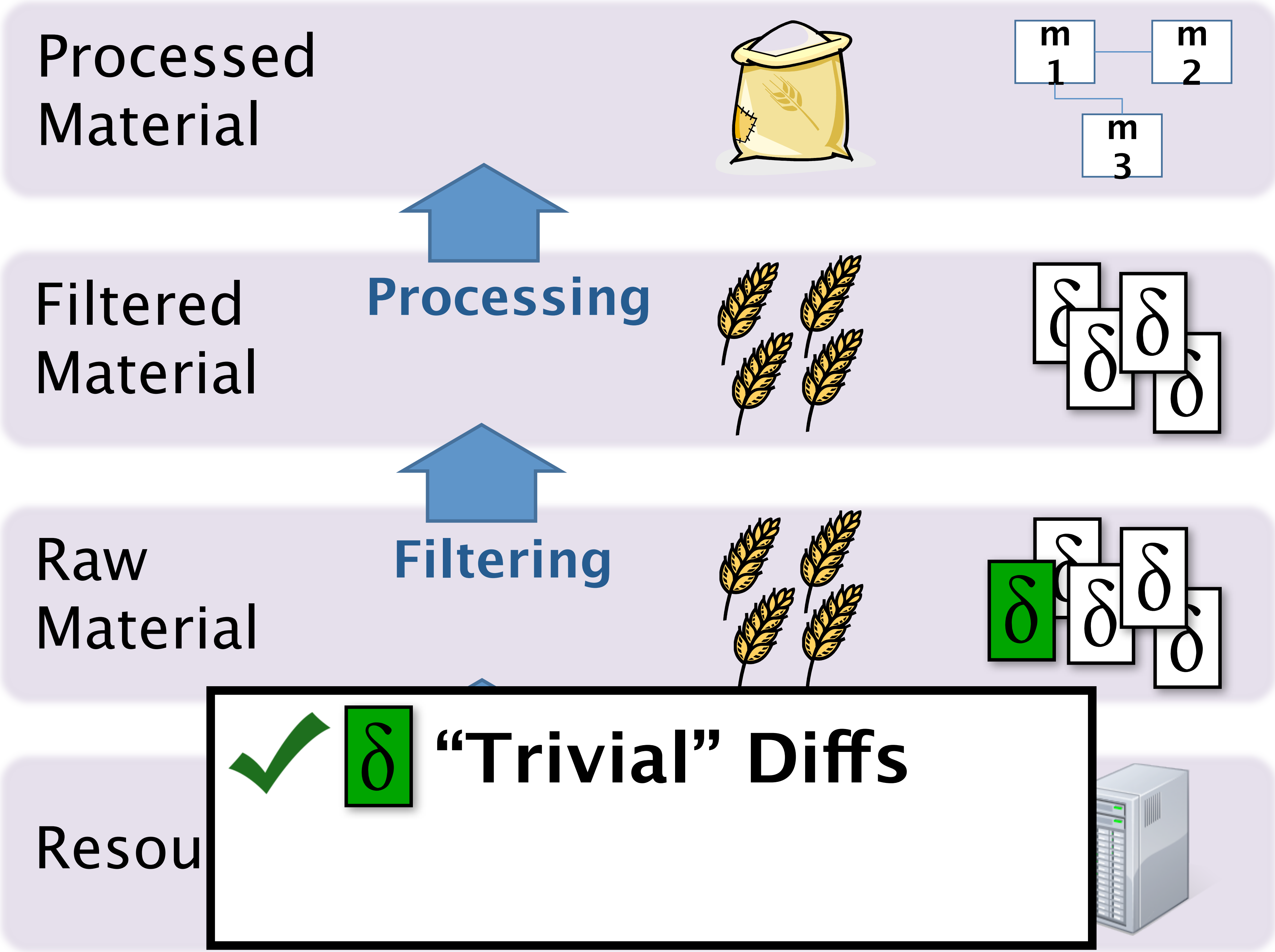

Finished
Product
$\frac{\{m_1, m_2\}}{m_3}$
Processed
Inference
m 1
m 2
m 3
잘못된 정보는 원하지
않는 결과를 초래
Material
δ
δ
δ
δ
Resource
Extraction


Finished Product
{m1, m2} / m3
Processed
Inference
m 1
m 2
m 3
잘못된 정보는 원하지 않는 결과를 초래
Material
δ
Extraction
Resource


Finished
Product
{m1, m2}
m3
Processed
Inference
m
1
m
2
m
3
잘못된 정보는 원하지
않는 결과를 초래
Material
δ
δ
δ
δ
Extraction
Resource


Finished Product
{m1, m2}
m3
Inference
m 1
m 2
m 3
잘못된 정보는 원하지 않는 결과를 초래
Material
δ
Extraction
Resource


Processed Material
m 1
m 2
m 3
Processing
Filtered Material
Filtering
Raw Material
Extraction
Resource


Processed
Material
m 1
m 2
m 3
Filtered
Material
Processing
δ
Raw
Material
Filtering
δ
Resou
δ
"Trivial" Diffs

뻔한 차이

Version N

```
Object field = …

void sample() {
   List l = …
   l.add(this.field);
   m(l.size());
   return;
}
```

Version N+1

```
void sample()
{
   List l = …
   l.add(this.field);
   m(l.size());
   return;
}

Object field = …
```

뻔한 차이

Version N

```
Object field = …

void sample() {
   List l = …
   l.add(this.field);
   m(l.size());
   return;
}
```

Version N+1

```
void sample()
{
   List l = …
   l.add(this.field);
   m(l.size());
   return;
}

Object field = …
```

뻔한 차이

Version N

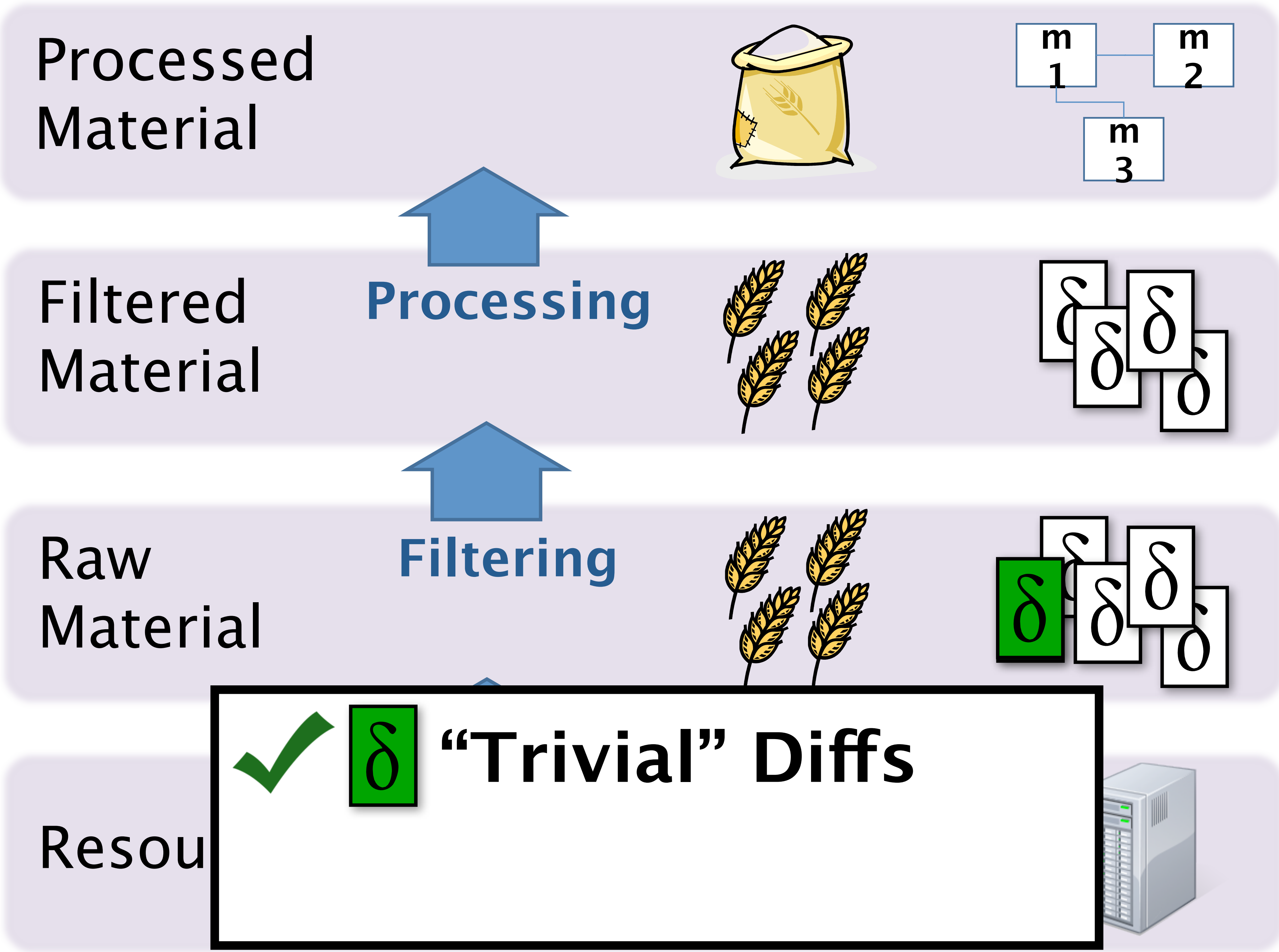
```
Object field = …

void sample() {
    List l = …
    l.add(this.field);
    m(l.size());
    return;
}
```

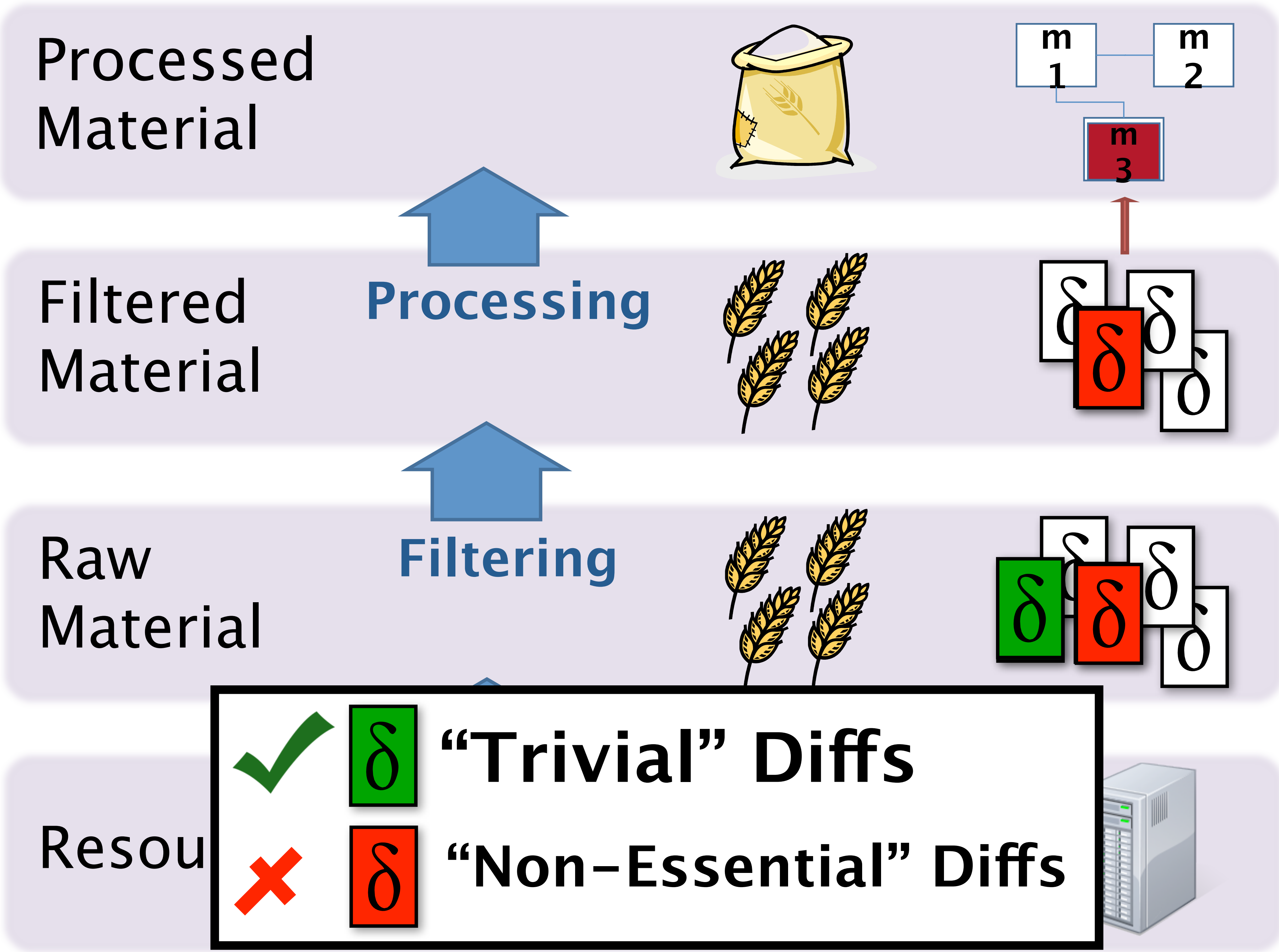
Version N+1

```
void sample()
{
    List l = …
    l.add(this.field);
    m(l.size());
    return;
}

Object field = …
```



Processed Material
m 1
m 2
m 3
Processing
Filtered Material
δ
Filtering
Raw Material
Resou
δ “Trivial” Diffs


Processed Material
m 1
m 2
m 3
Filtered Material
Processing
δ
Raw Material
Filtering
δ
Resou
δ
"Trivial" Diffs
δ
"Non-Essential" Diffs

불필요한 변경

Version N

```
Object field = …

void sample()
{
   List l = …
   l.add(this.field);
   m(l.size());
   return;
}
```

Version N+1

```
Object m_field = …

void sample()
{
   java.util.List list = …
   list.add(m_field);
   int size = list.size();
   m(size);
}
```

불필요한 변경

Version N

```
Object field = …

void sample()
{
   List l = …
   l.add(this.field);
   m(l.size());
   return;
}
```

Version N+1

```
Object m_field = …

void sample()
{
   java.util.List list = …
   list.add(m_field);
   int size = list.size();
   m(size);
}
```

불필요한 변경

Version N

```
Object field = …

void sample()
{
    List l = …
    l.add(this.field);
    m(l.size());
    return;
}
```

Version N+1

```
Object m_field = …

void sample()
{
    java.util.List list = …
    list.add(m_field);
    int size = list.size();
    m(size);
}
```

이름 바꾸기 때문에 생기는 변경

불필요한 변경

Version N

```
Object field = …

void sample()
{
    List l = …
    l.add(this.field);
    m(l.size());
    return;
}
```

Version N+1

```
Object m_field = …

void sample()
{
    java.util.List list = …
    list.add(m_field);
    int size = list.size();
    m(size);
}
```

이름 바꾸기 때문에 생기는 변경

간단한 키워드 변경

불필요한 변경

Version N

```
Object field = …

void sample()
{
    List l = …
    l.add(this.field);
    m(l.size());
    return;
}
```

이름 바꾸기 때문에 생기는 변경

간단한 키워드 변경

Version N+1

```
Object m_field = …

void sample()
{
    java.util.List list = …
    list.add(m_field);
    int size = list.size();
    m(size);
}
```

간단한 타입 변경

불필요한 변경

Version N

```
Object field = …

void sample()
{
    List l = …
    l.add(this.field);
    m(l.size());
    return;
}
```

이름 바꾸기 때문에 생기는 변경

간단한 키워드 변경

Version N+1

```
Object m_field = …

void sample()
{
    java.util.List list = …
    list.add(m_field);
    int size = list.size();
    m(size);
}
```

간단한 타입 변경

지역 변수 이름 바꾸기

불필요한 변경

Version N

```
Object field = …

void sample()
{
    List l = …
    l.add(this.field);
    m(l.size());
    return;
}
```

이름 바꾸기 때문에 생기는 변경

간단한 키워드 변경

임시 변수 도입으로 생기는 변경

Version N+1

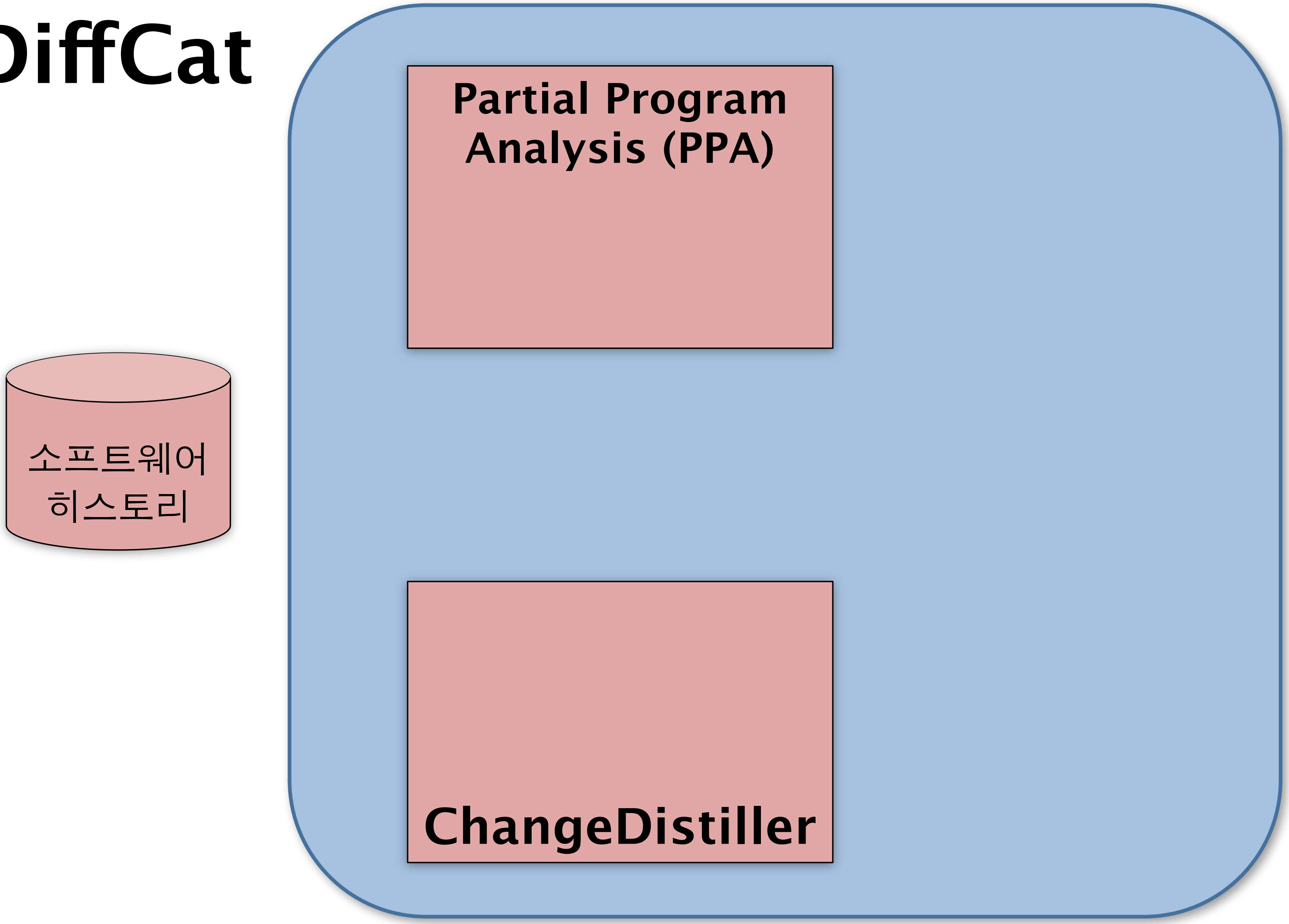
```
Object m_field = …

void sample()
{
    java.util.List list = …
    list.add(m_field);
    int size = list.size();
    m(size);
}
```

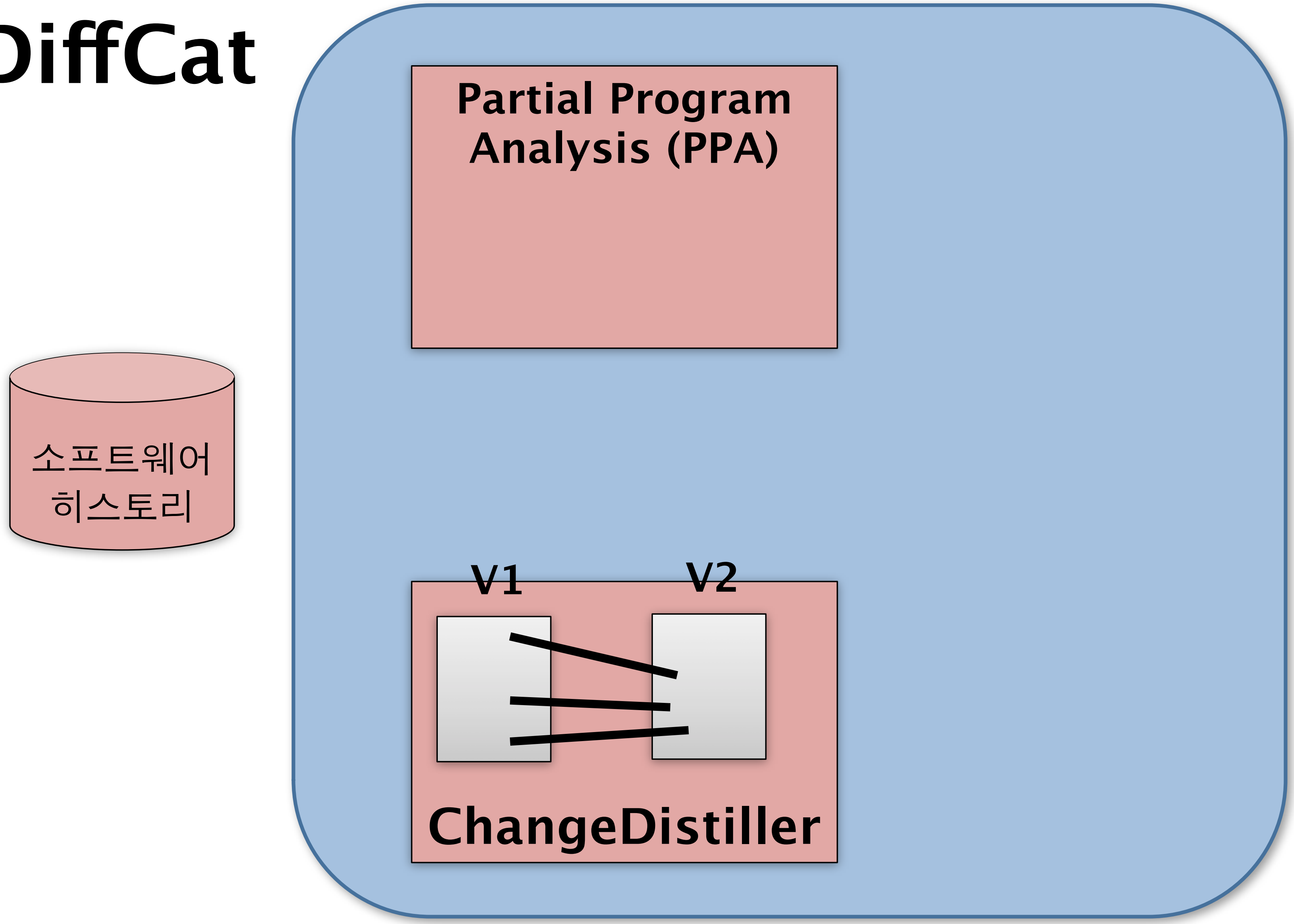
간단한 타입 변경

지역 변수 이름 바꾸기

DiffCat

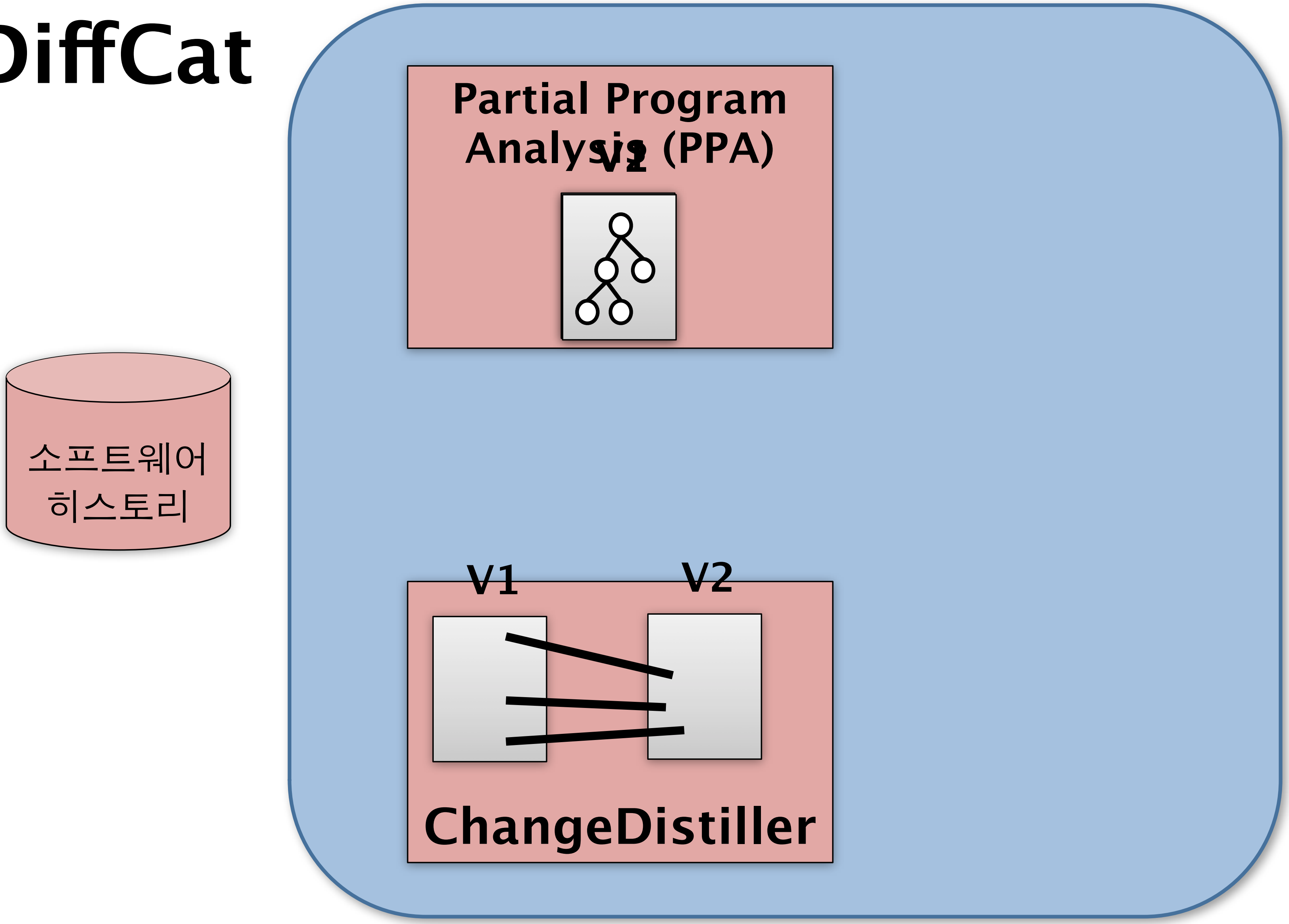


DiffCat

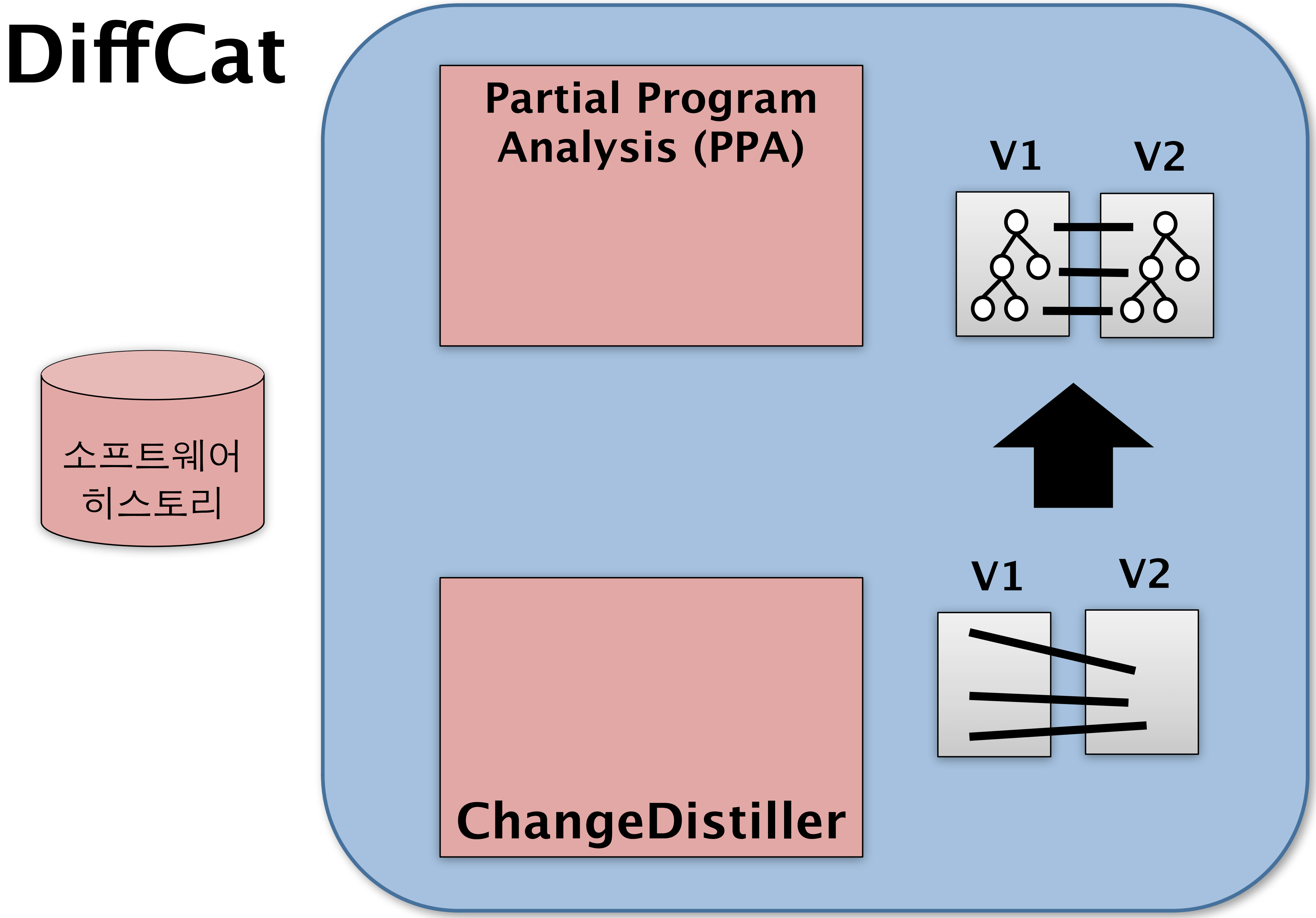


Partial Program Analysis (PPA)
소프트웨어 히스토리
V1
V2
ChangeDistiller

DiffCat



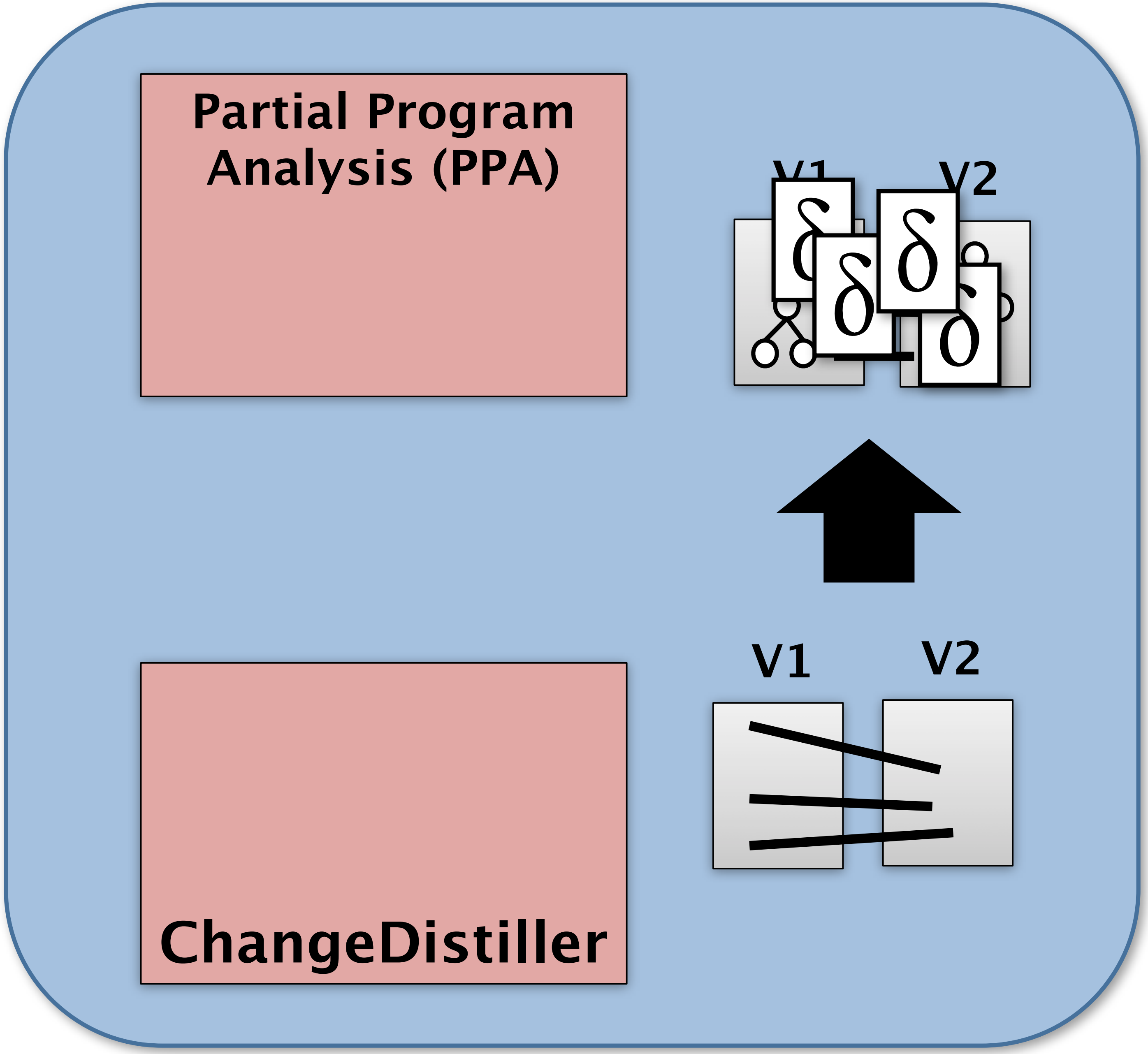
소프트웨어
히스토리
Partial Program
Analysis (PPA)
V1
V2
ChangeDistiller



DiffCat
Partial Program Analysis (PPA)
V1
V2
소프트웨어 히스토리
V1
V2
ChangeDistiller

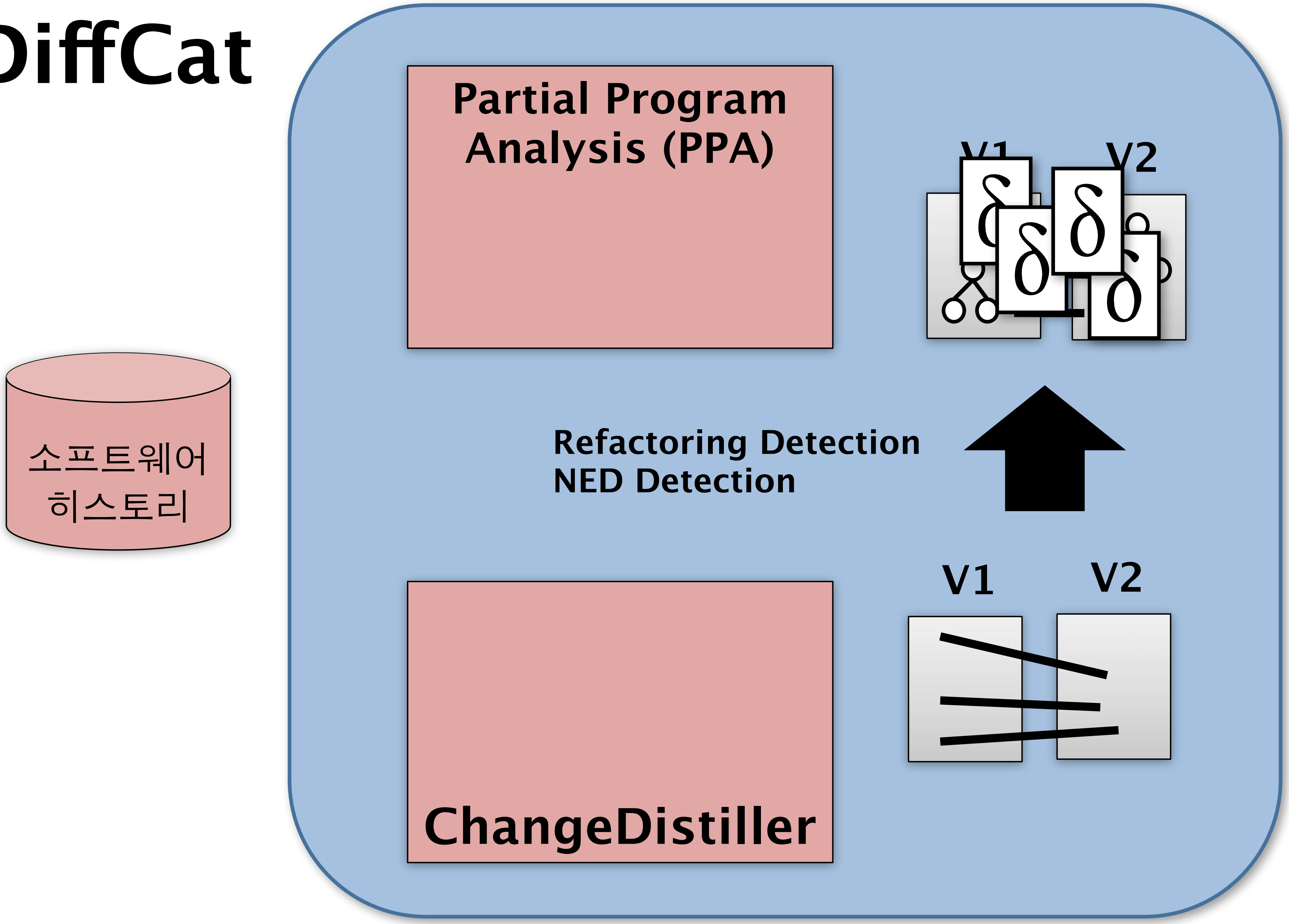
DiffCat

소프트웨어
히스토리

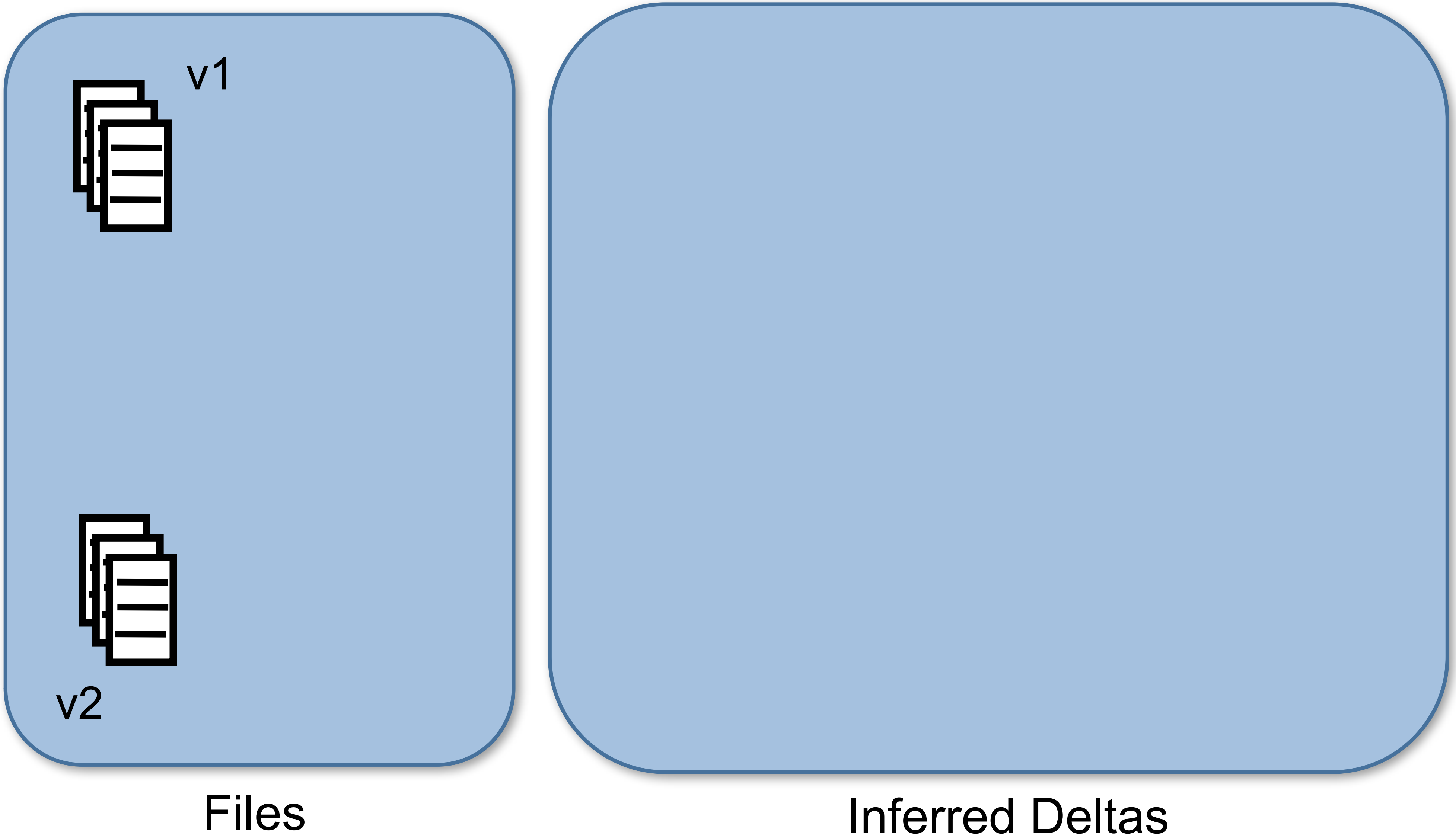


Partial Program
Analysis (PPA)
V2
V1
V2
ChangeDistiller

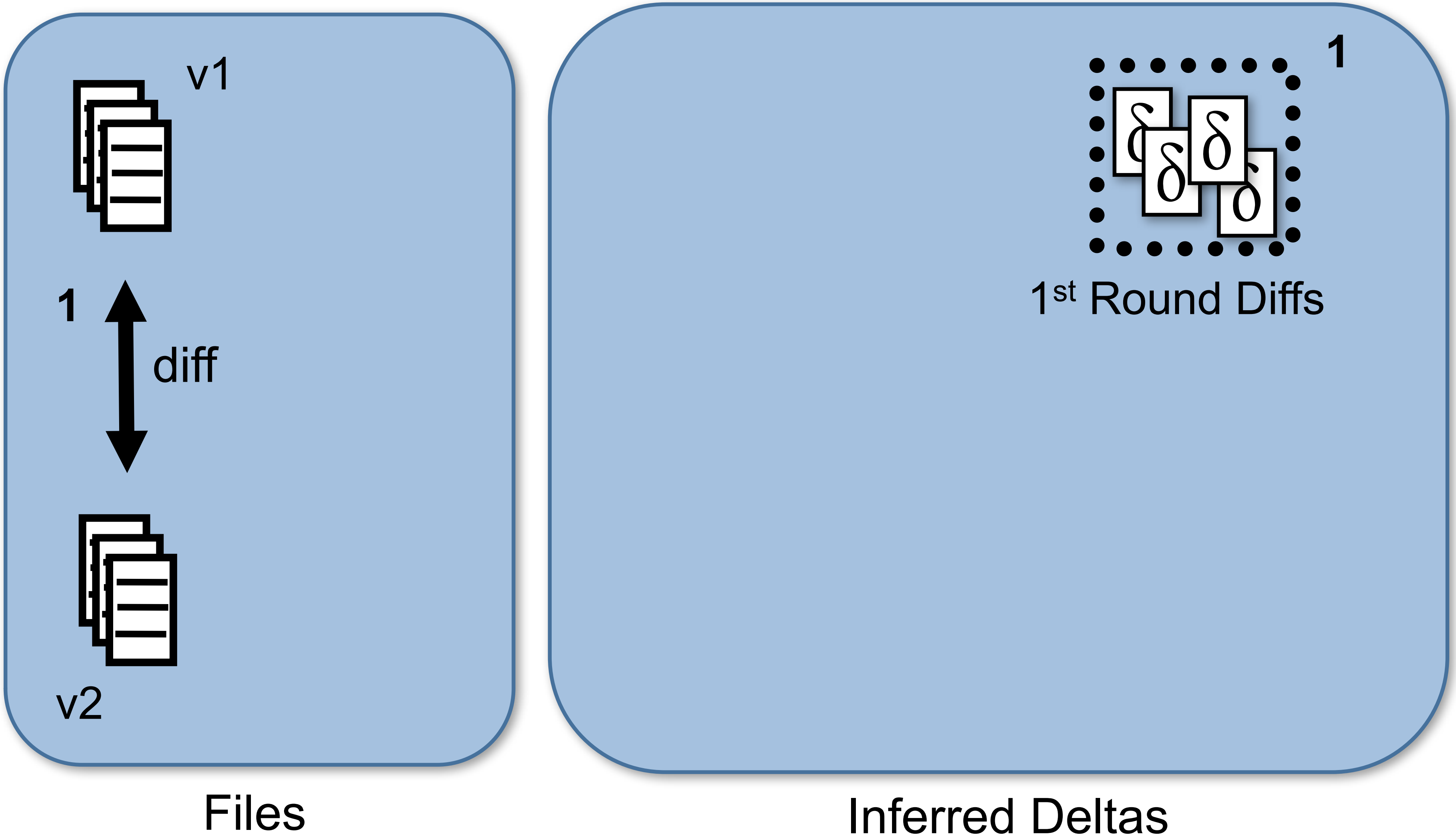
DiffCat


소프트웨어
히스토리
Partial Program
Analysis (PPA)
V2
Refactoring Detection
NED Detection
V1
V2
ChangeDistiller

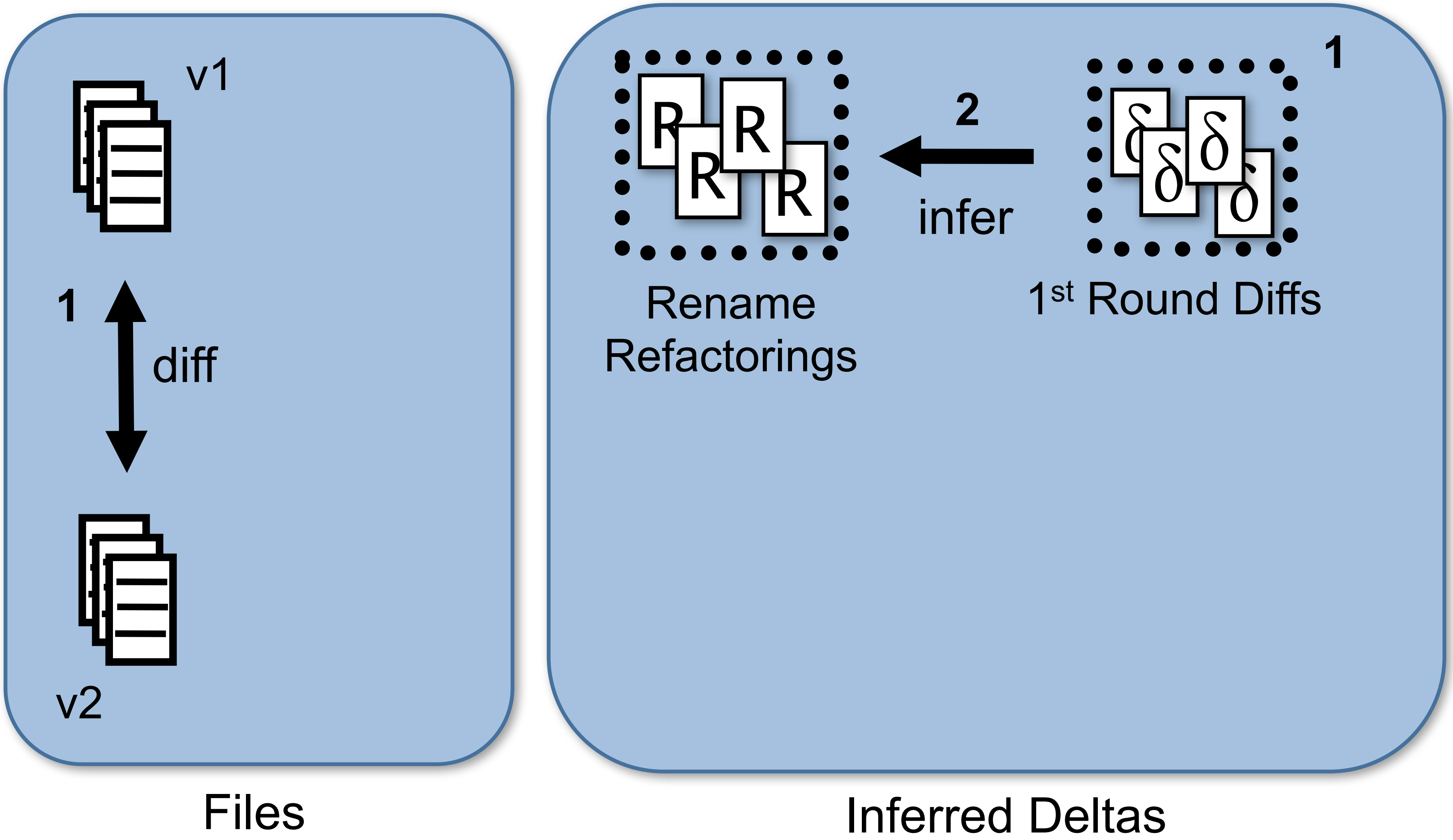
DiffCat



DiffCat

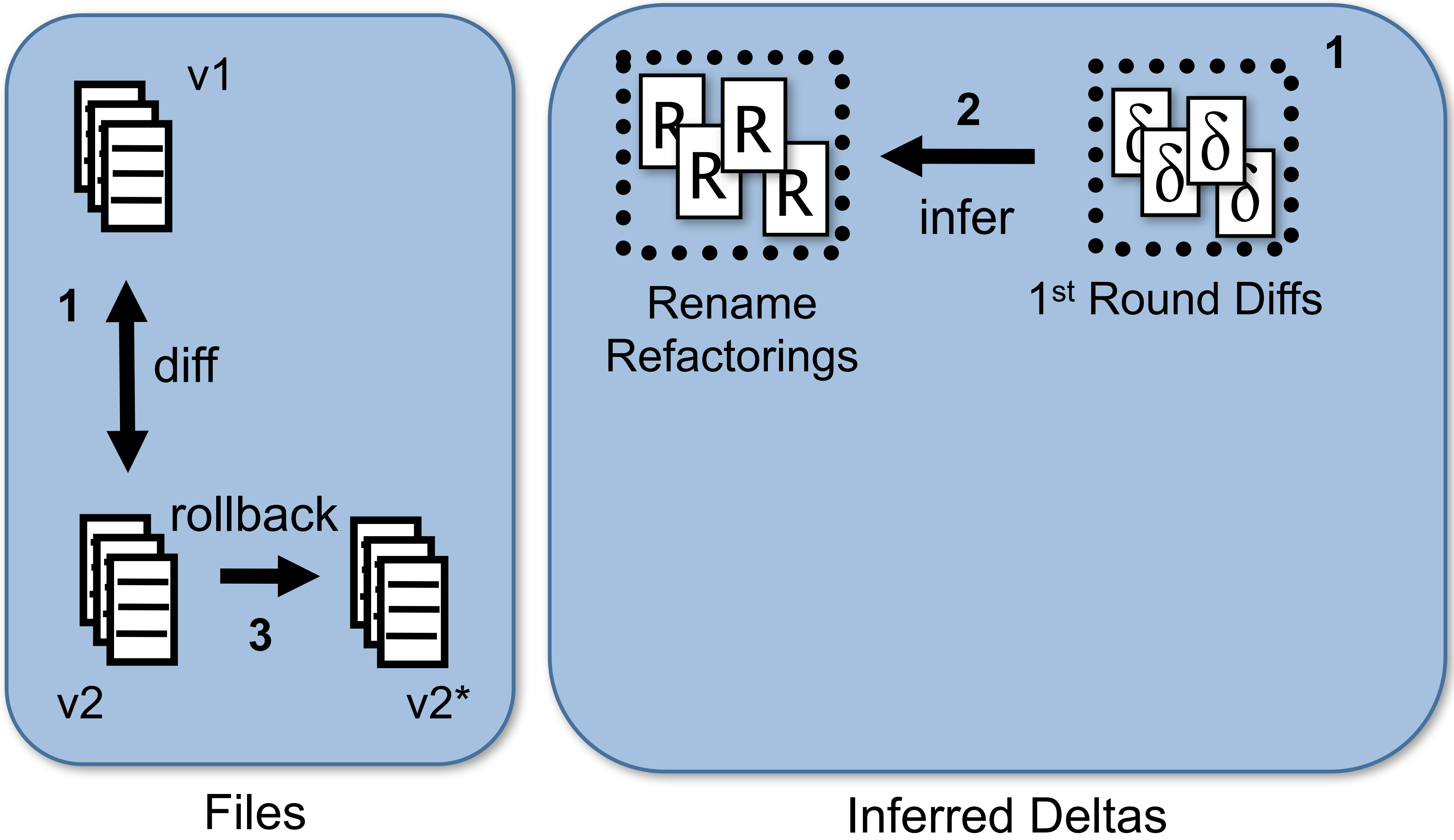


DiffCat

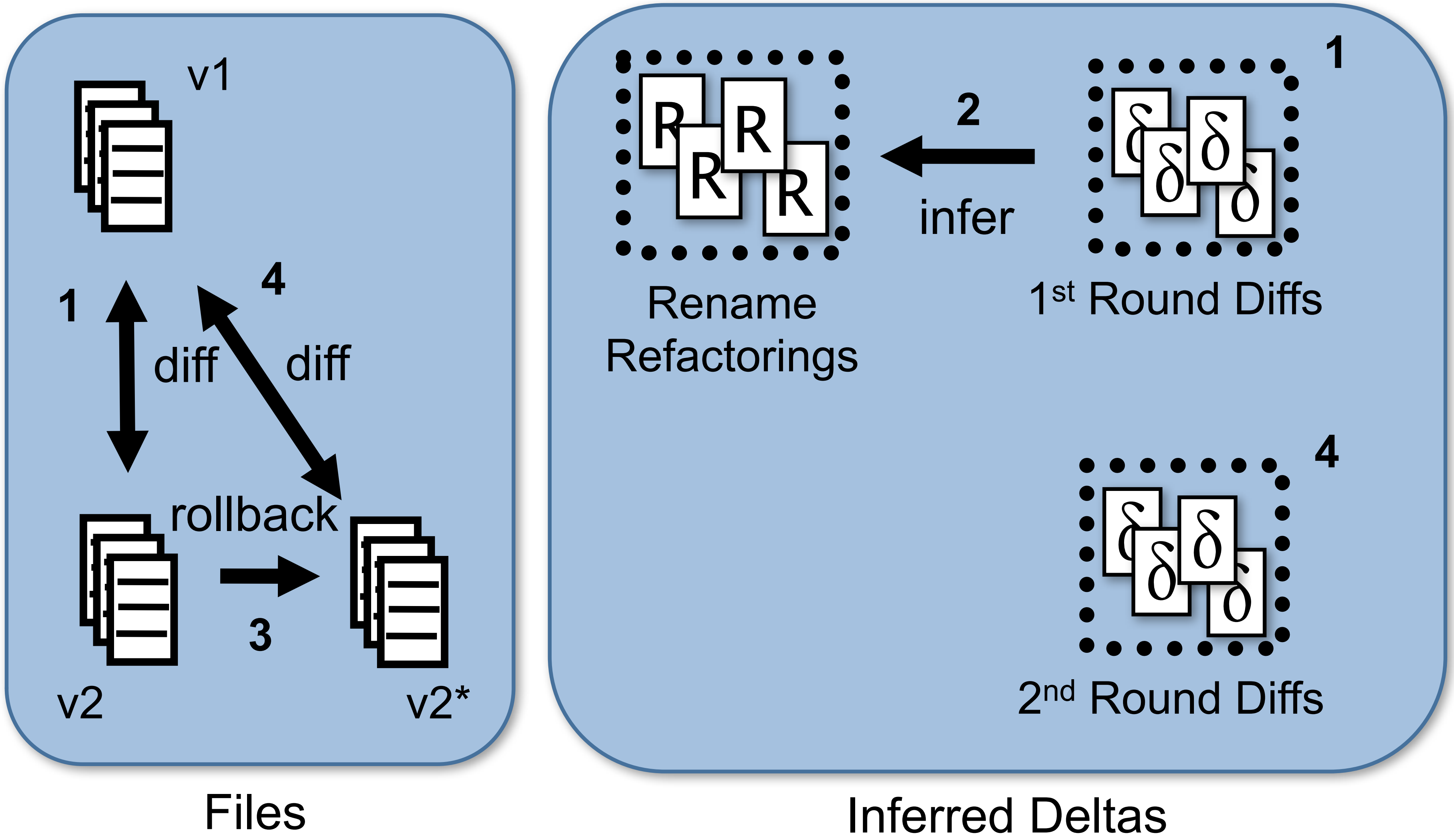


v1
1
diff
v2
Files
1
2
infer
Rename
Refactorings
1st Round Diffs
Inferred Deltas

DiffCat


v1
1
diff
rollback
3
v2
v2*
Files
R
2
infer
δ
1
Rename Refactorings
1st Round Diffs
Inferred Deltas

DiffCat



v1
1
diff
4
diff
rollback
3
v2
v2*
Files
1
2
infer
Rename Refactorings
1st Round Diffs
4
2nd Round Diffs
Inferred Deltas

예제

Version N

```
Object field = …

void sample() {
   field.foo( );
   List l = …
   l.add(this.field);
   m(l.size());
}
```

Version N+1

```
Object m_field = …

void sample() {
   m_field.foo( );
   List list = …
   list.add(m_field);
   int size = list.size();
   m(size);
}
```

예제

Version N

```
Object field = …

void sample() {
    field.foo( );
    List l = …
    l.add(this.field);
    m(l.size());
}
```

Version N+1

```
Object m_field = …

void sample() {
    m_field.foo( );
    List list = …
    list.add(m_field);
  + int size = list.size();
    m(size);
}
```

변경찾기

예제

Version N

```
Object field = …

void sample() {
    field.foo( );
    List l = …
    l.add(this.field);
    m(l.size());
}
```

Version N+1

```
Object m_field = …

void sample() {
    m_field.foo( );
    List list = …
    list.add(m_field);
    int size = list.size();
+   m(size);
}
```

변경찾기 이름바꾸기

예제

Version N

```
Object field = …

void sample() {
    field.foo( );
    List l = …
    l.add(this.field);
    m(l.size());
}
```

Version N+1

```
Object m_field = …

void sample() {
    m_field.foo( );
    List list = …
    list.add(m_field);
    int size = list.size();
    m(size);
}
```

예제

Version N

```
Object field = …

void sample() {
   field.foo( );
   List l = …
   l.add(this.field);
   m(l.size());
}
```

Version N+1*

```
Object field = …

void sample() {
   field.foo( );
   List l = …
   l.add(field);
   int size = l.size();
   m(size);
}
```

예제

Version N

```
Object field = …

void sample() {
   field.foo( );
   List l = …
   l.add(this.field);
   m(l.size());
}
```

Version N+1*

```
Object field = …

void sample() {
   field.foo( );
   List l = …
   l.add(field);
   int size = l.size();
   m(size);
}
```

예제

Version N

```
Object field = …

void sample() {
    field.foo( );
    List l = …
    l.add(this.field);
    m(l.size());
}
```

Version N+1*

```
Object field = …

void sample() {
    field.foo( );
    List l = …
    l.add(field);
  + int size = l.size();
    m(size);
}
```

변경찾기

예제

Version N

```
Object field = …

void sample() {
    field.foo( );
    List l = …
    l.add(this.field);
  - m(l.size());
}
```

Version N+1*

```
Object field = …

void sample() {
    field.foo( );
    List l = …
    l.add(field);
  + int size = l.size();
    m(size);
}
```

변경찾기

필요없는 this 없애기

예제

Version N

```
Object field = …

void sample() {
    field.foo( );
    List l = …
    l.add(this.field);
  - m(l.size());
}
```

Version N+1*

```
Object field = …

void sample() {
    field.foo( );
    List l = …
    l.add(field);
    int size = l.size();
    m(size);
}
```

변경찾기

필요없는 this 없애기

임시 변수 사용

예제

Version N

```
Object field = …

void sample() {
    field.foo( );
    List l = …
    l.add(this.field);
  - m(l.size());
}
```

Version N+1*

```
Object field = …

void sample() {
    field.foo( );
    List l = …
    l.add(field);
    int size = l.size();
    m(size);
}
```

변경찾기 | 필요없는 this 없애기 | 임시 변수 사용 | 이름변경

예제

Version N

```
Object field = …

void sample() {
   field.foo( );
   List l = …
   l.add(this.field);
 - m(l.size());
}
```

Version N+1*

```
Object field = …

void sample() {
   field.foo( );
   List l = …
   l.add(field);
   int size = l.size();
   m(size);
}
```

이름바꾸기 전파

변경찾기

필요없는 this 없애기

임시 변수 사용

이름변경

불필요한 메소드 변경이 얼마나 많나?

시스템	전체 함수	불필요한 함수
Ant	17 792	2 759 (15.5%)
Azureus	8 731	229 (2.6%)
Hibernate	15 881	1 153 (7.3%)
JDT-Core	8 837	673 (7.6%)
JDT-UI	9 681	426 (4.4%)
Spring Framewrk	11 047	1 715 (15.5%)
Xerces	8 409	256 (3.0%)
합계	80 378	7 211 (9.0%)

불필요한 메소드 변경이 얼마나 많나?

시스템	전체 함수	불필요한 함수
Ant	17 792	2 759 (15.5%)
Azureus	8 731	229 (2.6%)
Hibernate	15 881	1 153 (7.3%)
JDT-Core	8 837	673 (7.6%)
JDT-UI	9 681	426 (4.4%)
Spring Framewrk	11 047	1 715 (15.5%)
Xerces	8 409	256 (3.0%)
합계	80 378	7 211 (9.0%)

불필요한 메소드 변경이 얼마나 많나?

시스템	전체 함수	불필요한 함수
Ant	17 792	2 759 (15.5%)
Azureus	8 731	229 (2.6%)
Hibernate	15 881	1 153 (7.3%)
JDT-Core	8 837	673 (7.6%)
JDT-UI	9 681	426 (4.4%)
Spring Framewrk	11 047	1 715 (15.5%)
Xerces	8 409	256 (3.0%)
합계	80 378	7 211 (9.0%)

불필요한 변경 제거하면 얼마나 좋나?

Setup	Tot Rec	True Rec	Prec	Top 3	Only Err
원래대로	93 576	20 501	0.219	0.442	0.220
불필요한 변경 제거	81 162	19 631	0.242	0.475	0.183
Δ					

$$\frac{0.242}{0.219} = 1.105$$

불필요한 변경 제거하면 얼마나 좋나?

Setup	Tot Rec	True Rec	Prec	Top 3	Only Err
원래대로	93 576	20 501	0.219	0.442	0.220
불필요한 변경 제거	81 162	19 631	0.242	0.475	0.183
Δ			+10.5%	+7.5%	-20.2%

$$\frac{0.242}{0.219} = 1.105$$

불필요한 변경 제거하면 얼마나 좋나?

Setup	Tot Rec	True Rec	Prec	Top 3	Only Err
원래대로	93 576	20 501	0.219	0.442	0.220
불필요한 변경 제거	81 162	19 631	0.242	0.475	0.183
Δ	-13.3%	-4.6%	+10.5%	+7.5%	-20.2%

$$\frac{0.242}{0.219} = 1.105$$

불필요한 변경 제거하면 얼마나 좋나?

Setup	Tot Rec	True Rec	Prec	Top 3	Only Err
원래대로	93 576	20 501	0.219	0.442	0.220
불필요한 변경 제거	81 162	19 631	0.242	0.475	0.183
Δ	-13.3%	-4.6%	+10.5%	+7.5%	-20.2%

$$\frac{0.242}{0.219} = 1.105$$

정적 분석 기술로 소프트웨어 공학의 문제를 풀어보자!

의미 코드 쌍 찾기

MeCC: Memory Comparison-based Clone Detector*

Heejung Kim†, Yungbum Jung†, Sunghun Kim§, Kwangkeun Yi†
†Seoul National University, Seoul, Korea
{hjkim,dreameye,kwang}@ropas.snu.ac.kr
§The Hong Kong University of Science and Technology, Hong Kong
hunkim@cse.ust.hk

ABSTRACT

In this paper, we propose a new semantic clone detection technique by comparing programs' abstract memory states, which are computed by a semantic-based static analyzer. Our experimental study using three large-scale open source projects shows that our technique can detect semantic clones that existing syntactic- or semantic-based clone detectors miss. Our technique can help developers identify inconsistent clone changes, find refactoring candidates, and understand software evolution related to semantic clones.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*restructuring, reverse engineering, and reengineering*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*program analysis*

General Terms

Languages, Algorithms, Experimentation

Keywords

Clone detection, abstract interpretation, static analysis, software maintenance

1. INTRODUCTION

Detecting code clones is useful for software development and maintenance tasks including identifying refactoring candidates [11], finding potential bugs [16, 15], and understanding software evolution [20, 6].

Most clone detectors [13, 19, 24, 9, 22] are based on textual similarity. For example, CCFinder [19] extracts and compares textual tokens from source code to determine code clones. Deckard [13] compares characteristic vectors extracted from abstract syntax trees (ASTs).

Although these detectors are good at detecting syntactic clones, they are not effective to detect semantic clones that are functionally similar but syntactically different.

A few existing approaches to detect semantic clones (e.g., those based on program dependence graphs (PDGs)[22, 9, 25] or by observing program executions via random testing [14]) have limitations. PDGs can be affected by syntactic changes such as replacing statements with a semantically equivalent procedure call. Hence, the PDG-based clone detectors miss some semantic clones. The clone detectability of random testing-based approaches may depend on the limited test coverage, covering only up to 60 ∼ 70% of software [27, 28, 35].

To detect semantic clones effectively, we propose a new clone detection technique: (1) we first use a path-sensitive semantic-based static analyzer to estimate the memory states at each procedure's exit point; (2) then we compare the memory states to determine clones. Since the abstract memory states have a collection of the memory effects (though approximated) along the execution paths within procedures, our technique can effectively detect semantic clones, and our clone detection ability is independent of syntactic similarity of clone candidates.

We implemented our technique as a clone detection tool, Memory Comparison-based Clone detector (MeCC), by extending a semantic-based static analyzer[18, 17, 12]. The extension is to support path-sensitivity and record abstract memory states. Our experiments with three large-scale open source projects, Python, Apache, and PostgreSQL (Section 4) show that MeCC can identify semantic clones that other existing methods miss.

The semantic clones identified by MeCC can be used for software development and maintenance tasks such as identifying refactoring candidates, detecting inconsistencies for locating potential bugs, and detecting software plagiarism (as discussed in Section 5.1).

This paper makes the following contributions:

- **Abstract memory-based clone detection technique:** We show that using abstract memory states that are computed by semantic-based static analysis is effective to detect semantic clones.
- **Semantic clone detector MeCC:** We implemented the proposed technique as a tool, MeCC (`http://ropas.`

*This work was supported by the Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology (MEST) / National Research Foundation of Korea (NRF) (Grant 2010-0001717). This work was partly supported by (A) the Brain Korea 21 Project, School of Electrical Engineering and Computer Science, Seoul National University, (B) Fasoo.com, and (C) Samsung Electronics.

ICSE '11, May 21–28, 2011, Honolulu, Hawaii, USA


```
PyObject *PyBool_FromLong (long ok) {
    PyObject *result;
    if (ok) result = Py_True;
    else result = Py_False;
    Py_INCREF(result);
    return result;
}
```

```
static PyObject *get_pybool (int istrue)
{
    PyObject *result =
      istrue? Py_True: Py_False;

    Py_INCREF(result);
    return result;
}
```

```
... *set_access_name(cmd_parms *cmd, void *dummy, const char *arg){
  void *sconf = cmd->server->module_config;
  core_server_config *conf =
      ap_get_module_config(sconf, &core_module);
  const char *err = ap_check_cmd_context(sconf,NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
  if (err != NULL) {
      return err;
   }
   conf->access_name = apr_pstrdup(cmd->pool,arg);
   return NULL;
}
```

```
... *set_protocol(cmd_parms *cmd, void *dummy, const char *arg){
  const char *err = ap_check_cmd_context(cmd,NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
  core_server_config *conf =
      ap_get_module_config(cmd->server->module_config, &core_module);
  char *proto;

  if (err != NULL) {
      return err;
   }
   proto = apr_pstrdup(cmd->pool,arg);
   ap_str_tolower(proto);
   conf->protocol = proto;
   return NULL;
}
```

```
... *set_access_name(cmd_parms *cmd, void *dummy, const char *arg){
  void *sconf = cmd->server->module_config;
  core_server_config *conf =
      ap_get_module_config(sconf, &core_module);
  const char *err = ap_check_cmd_context(sconf,NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
  if (err != NULL) {
      return err;
   }
   conf->access_name = apr_pstrdup(cmd->pool,arg);
   return NULL;
}
```

statement reordering

```
... *set_protocol(cmd_parms *cmd, void *dummy, const char *arg){
  const char *err = ap_check_cmd_context(cmd,NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
  core_server_config *conf =
      ap_get_module_config(cmd->server->module_config, &core_module);
  char *proto;

  if (err != NULL) {
      return err;
   }
   proto = apr_pstrdup(cmd->pool,arg);
   ap_str_tolower(proto);
   conf->protocol = proto;
   return NULL;
}
```

```
... *set_access_name(cmd_parms *cmd, void *dummy, const char *arg){
  void *sconf = cmd->server->module_config;
  core_server_config *conf =
      ap_get_module_config(sconf, &core_module);
  const char *err = ap_check_cmd_context(sconf,NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
  if (err != NULL) {
      return err;
   }
   conf->access_name = apr_pstrdup(cmd->pool,arg);
   return NULL;
}
```

statement reordering

intermediate variables

```
... *set_protocol(cmd_parms *cmd, void *dummy, const char *arg){
  const char *err = ap_check_cmd_context(cmd,NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
  core_server_config *conf =
      ap_get_module_config(cmd->server->module_config, &core_module);
  char *proto;

  if (err != NULL) {
      return err;
   }
   proto = apr_pstrdup(cmd->pool,arg);
   ap_str_tolower(proto);
   conf->protocol = proto;
   return NULL;
}
```

```
... *set_access_name(cmd_parms *cmd, void *dummy, const char *arg){
  void *sconf = cmd->server->module_config;
  core_server_config *conf =
      ap_get_module_config(sconf, &core_module);
  const char *err = ap_check_cmd_context(sconf,NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
  if (err != NULL) {
      return err;
   }
   conf->access_name = apr_pstrdup(cmd->pool,arg);
   return NULL;
}
```

statement reordering

intermediate variables

statement splitting

```
... *set_protocol(cmd_parms *cmd, void *dummy, const char *arg){
  const char *err = ap_check_cmd_context(cmd,NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
  core_server_config *conf =
      ap_get_module_config(cmd->server->module_config, &core_module);
  char *proto;

  if (err != NULL) {
      return err;
   }
   proto = apr_pstrdup(cmd->pool,arg);
   ap_str_tolower(proto);
   conf->protocol = proto;
   return NULL;
}
```

```
... *set_access_name(cmd_parms *cmd, void *dummy, const char *arg){
  void *sconf = cmd->server->module_config;
  core_server_config *conf =
      ap_get_module_config(sconf, &core_module);
  const char *err = ap_check_cmd_context(sconf,NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
  if (err != NULL) {
      return err;
   }
   conf->access_name = apr_pstrdup(cmd->pool,arg);
   return NULL;
}
```

statement reordering | intermediate variables | statement splitting

```
... *set_protocol(cmd_parms *cmd, void *dummy, const char *arg){
  const char *err = ap_check_cmd_context(cmd,NOT_IN_DIR_LOC_FILE | NOT_IN_LIMIT);
  core_server_config *conf =
      ap_get_module_config(cmd->server->module_config, &core_module);
  char *proto;

  if (err != NULL) {
      return err;
   }
   proto = apr_pstrdup(cmd->pool,arg);
   ap_str_tolower(proto);
   conf->protocol = proto;
   return NULL;
}
```

불필요한 변경 예제

Version N

```
Object field = …

void sample() {
   field.foo( );
   List l = …
   l.add(this.field);
   m(l.size());
}
```

Version N+1

```
Object m_field = …

void sample() {
   m_field.foo( );
   List list = …
   list.add(m_field);
   int size = list.size();
   m(size);
}
```

불필요한 변경 예제

Version N

```
Object field = …

void sample() {
   field.foo( );
   List l = …
   l.add(this.field);
   m(l.size());
}
```

Version N+1

```
Object m_field = …

void sample() {
   m_field.foo( );
   List list = …
   list.add(m_field);
   int size = list.size();
   m(size);
}
```

보기에는 다를 수 있는데 하는 일은 같은 함수

불필요한 변경 예제

Version N

```
Object field = …

void sample() {
   field.foo( );
   List l = …
   l.add(this.field);
   m(l.size());
}
```

Version N+1

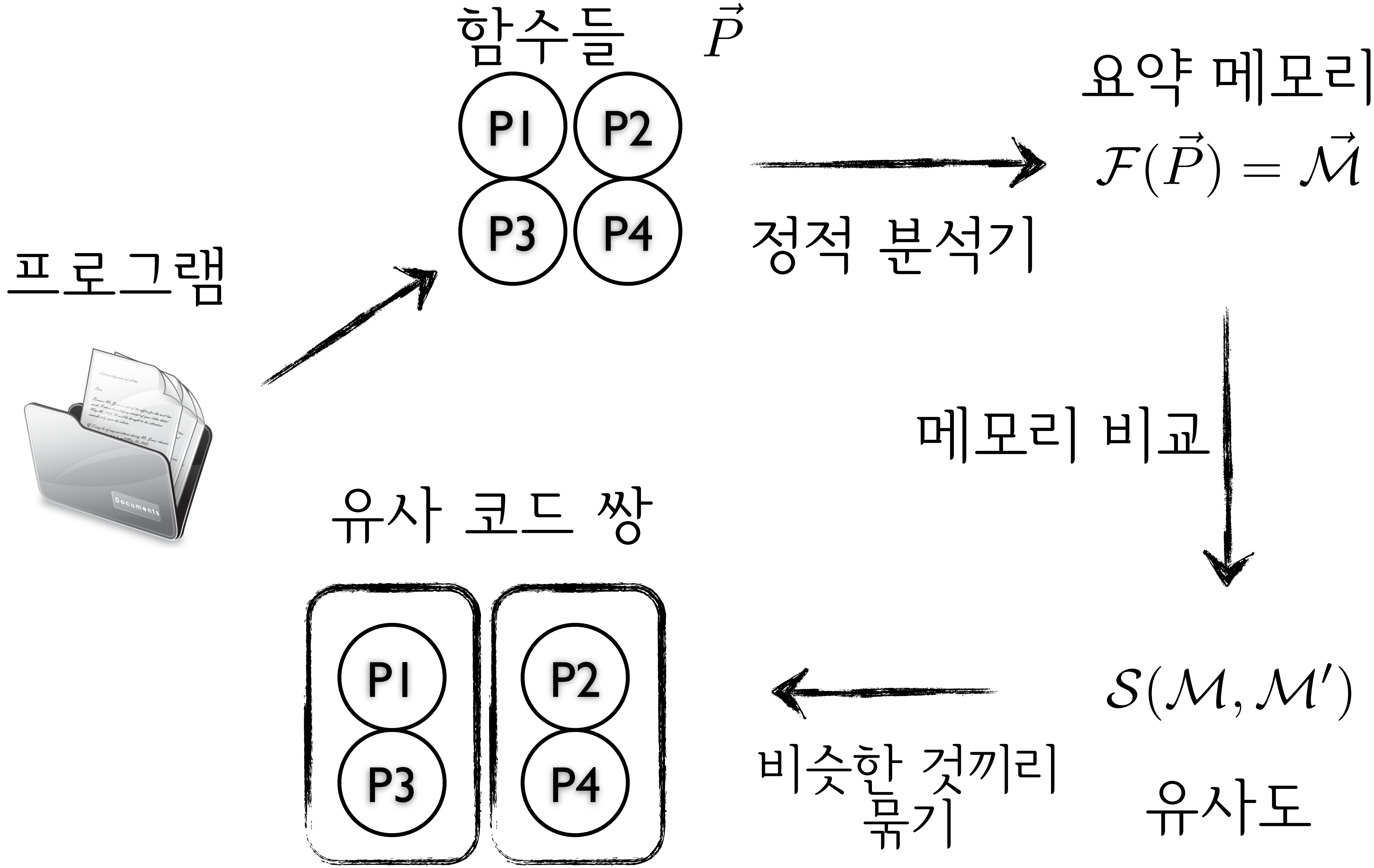
```
Object m_field = …

void sample() {
   m_field.foo( );
   List list = …
   list.add(m_field);
   int size = list.size();
   m(size);
}
```

보기에는 다를 수 있는데 하는 일은 같은 함수

의미 코드 쌍 (semantic code clone)

유사 코드 쌍 찾는 방법

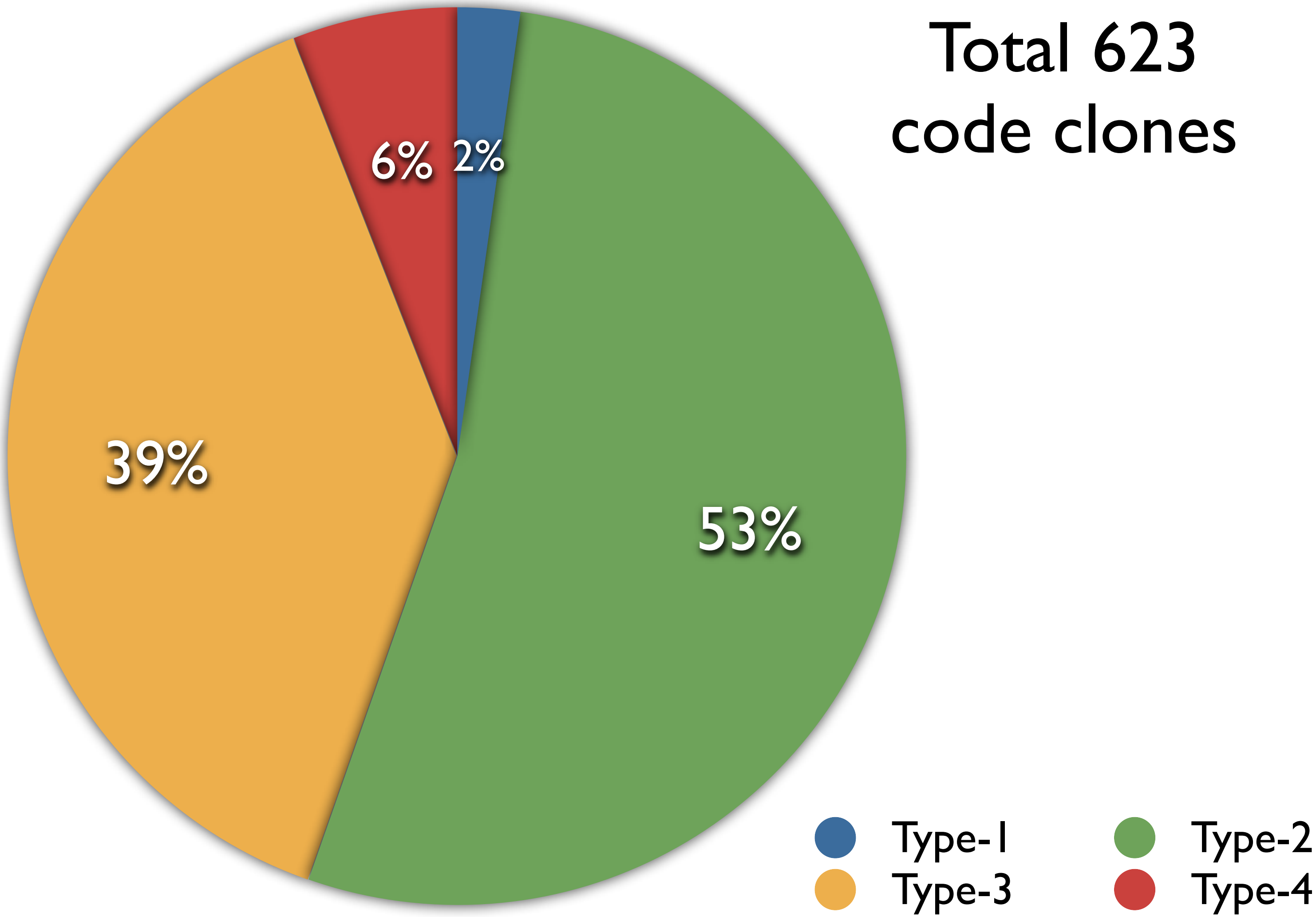


함수들 $\vec{P}$
P1
P2
P3
P4
요약 메모리
$\mathcal{F}(\vec{P}) = \vec{\mathcal{M}}$
정적 분석기
프로그램
Documents
메모리 비교
유사 코드 쌍
P1
P3
P2
P4
$\mathcal{S}(\mathcal{M}, \mathcal{M}')$
비슷한 것끼리 묶기
유사도

실험 대상

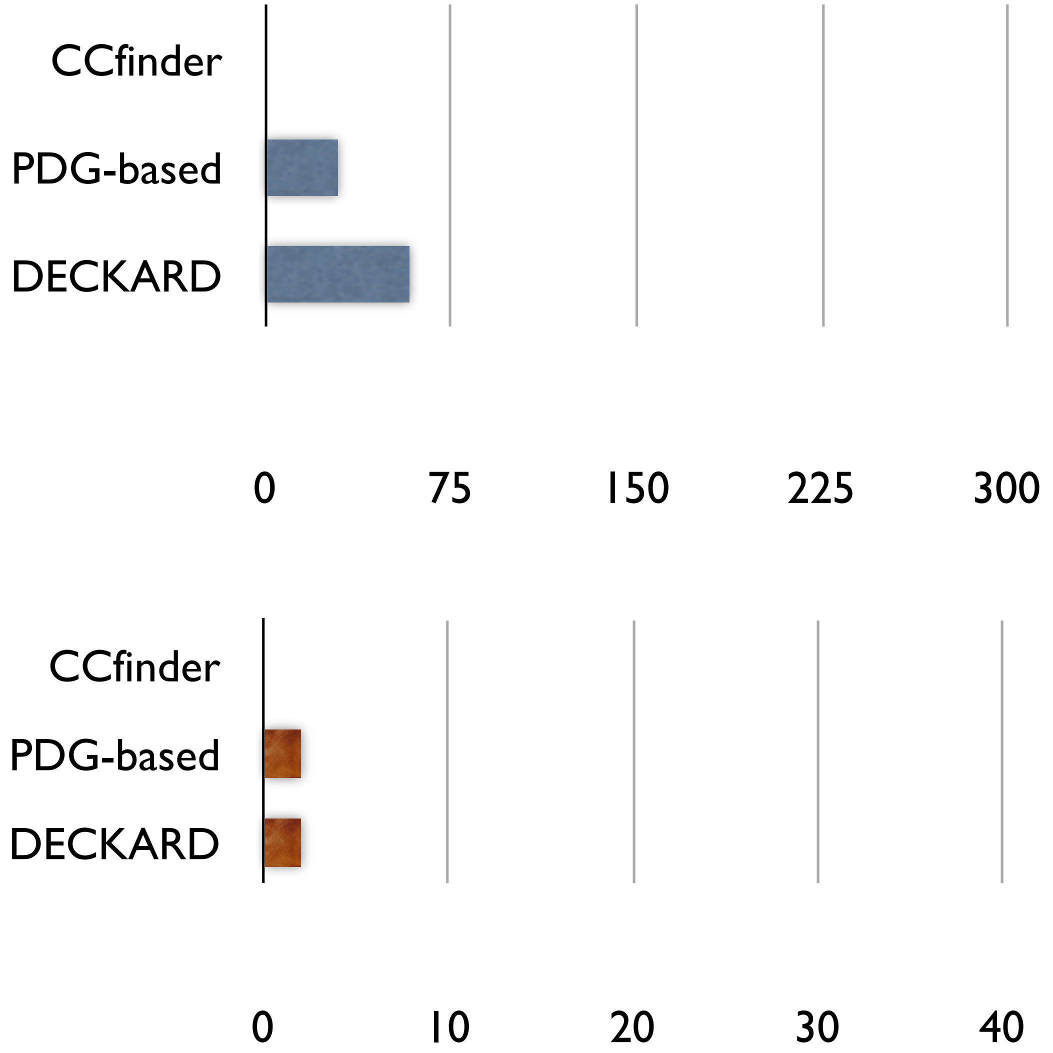
Projects	KLOC	Procedures	Application
Python	435	7,657	interpreter
Apache	343	9,483	web server
PostgreSQL	937	10,469	database

찾은 유사 코드 쌍



C. K. Roy and J. R. Cordy. A survey on software clone detection research. SCHOOL OF COMPUTING TR 2007-541, QUEEN'S UNIVERSITY, 115, 2007.

다른 도구들과 비교

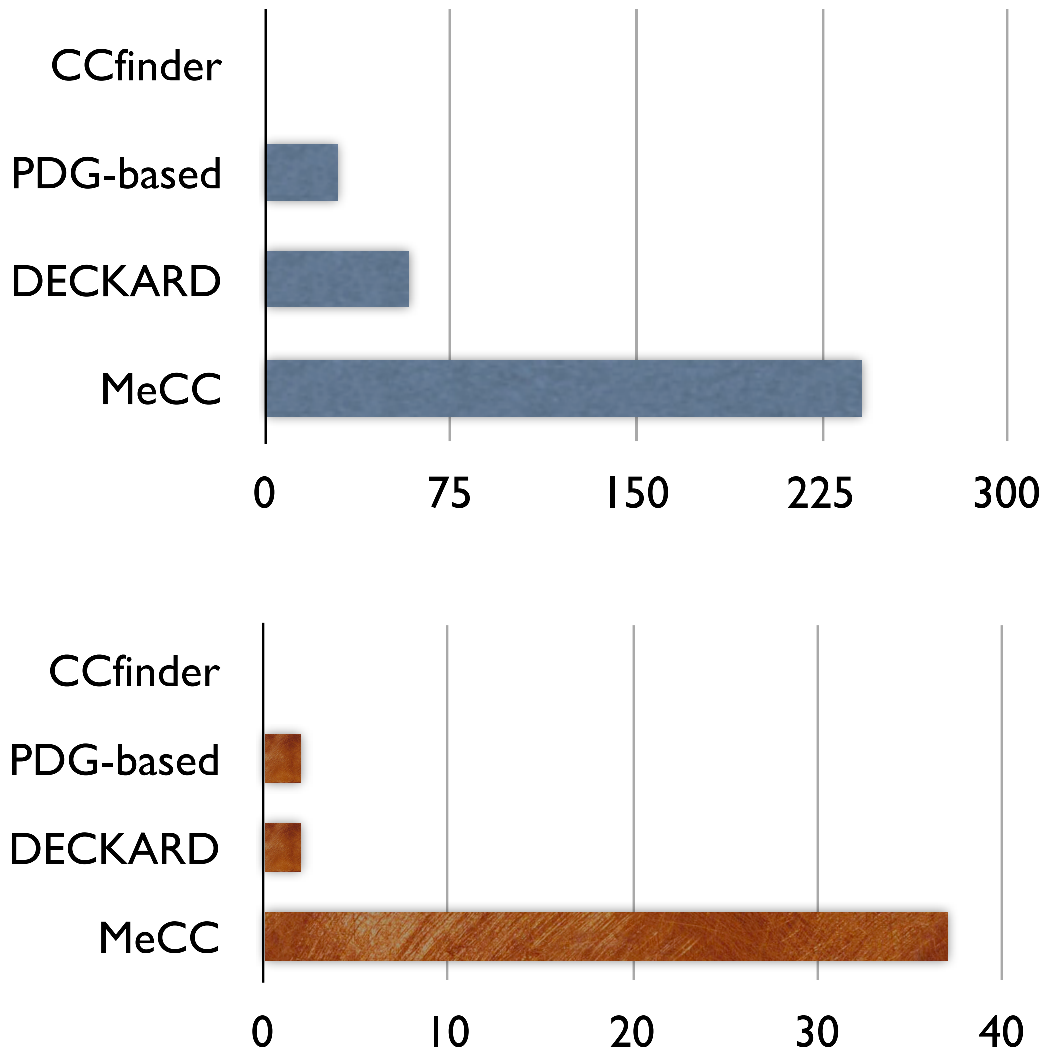


CCfinder
textual tokens

PDG-based
program dependency graphs

DECKARD
characteristic vectors

다른 도구들과 비교



CCfinder
textual tokens

PDG-based
program dependency graphs

DECKARD
characteristic vectors

문제점?

- DiffCat은 자바프로그램만, MeCC은 C 프로그램만 지원
- MeCC은 함수 단위로만 유사 코드 쌍을 찾음

문제점?

- DiffCat은 자바프로그램만, MeCC은 C 프로그램만 지원
- MeCC은 함수 단위로만 유사 코드 쌍을 찾음

자바 분석기 위에 MeCC 방법을 적용하는 구현 중

참고 자료

Non-Essential Changes in Version Histories

David Kawrykow and Martin P. Robillard
McGill University
Montréal, Canada
{dkawry,martin}@cs.mcgill.ca

ABSTRACT

Numerous techniques involve mining change data captured in software archives to assist engineering efforts, for example to identify components that tend to evolve together. We observed that important changes to software artifacts are sometimes accompanied by numerous *non-essential* modifications, such as local variable refactorings, or textual differences induced as part of rename refactoring. We developed a tool-supported technique for detecting non-essential code differences in the revision histories of software systems. We used our technique to investigate code changes in over 24 000 change sets gathered from the change histories of seven long-lived open-source systems. We found that up to 15.5% of a system's method updates were due solely to non-essential differences. We also report on numerous observations on the distribution of non-essential differences in change history and their potential impact on change-based analyses.

Keywords

Mining software repositories, software change analysis, differencing algorithms

1. INTRODUCTION

Source code repository systems have been in use since the 1970s to keep track of the different versions of a system's artifacts and, by extension, of the changes made between versions [22]. Numerous techniques now involve mining change data captured in software archives to assist software engineering efforts. For example, mining change data has been used to measure code decay in aging systems [5], to predict defects in modules [10, 16], and to detect non-obvious relationships between code elements [8, 23, 25]. We refer to approaches operating on change data as *change-based approaches*.

Typical version control systems store changes as line-based textual deltas between committed code files. In contrast, change-based approaches generally aim to operate on more meaningful representations of change, such as, for example, the individual methods that were updated as part of a developer commit to the repository. More meaningful representations of software changes support more accurate reasoning about software development activity and effort.

A critical problem for change-based approaches is thus to bridge this conceptual gap between the low-level deltas stored in version control systems and the abstractions used to represent software development activity. A first step, implemented by most modern change-based approaches, is to ignore trivial low-level changes, like those induced by white spaces or other formatting-related modifications. The general assumption behind this strategy is that these groups of low-level differences are less likely to yield meaningful abstractions of the actual development effort behind a code change. For example, many change-based approaches ignore trivial updates when determining the set of methods that were modified as part of a code commit.

As part of our ongoing investigation of software archives, we observed that many additional kinds of minor (or *non-essential*) code changes can also cause change-based approaches to infer inaccurate high-level representations of software development effort. For example, every time a developer performs a rename refactoring, all methods that include references to the renamed element will also be textually modified; a naive abstraction of these non-essential rename-induced statement updates can then result in a bloated high-level representation of the change that appears to span many lines of code, methods, and files, despite corresponding to a single developer modification (that is generally a very simple tool-assisted operation). Given the growing importance of change analysis in software engineering, our long-term goal is to enable change-based approaches to incorporate information about the *essentiality* of code changes into their analyses. With this information, change-based approaches will be able to more precisely select the individual low-level modifications they use to derive their high-level representations of development activity or effort.

We investigated the potential impact of non-essential differences on the abstractions that are typically analyzed by many change-based approaches. In particular, we sought *i)* to characterize the prevalence of non-essential differences in change history, and, *ii)* to measure their impact on the *code churn* and *method updates* associated with code commits, two facets of code change that are considered in existing empirical studies involving change data [5, 16] and change task oriented analyses [25].

Analyzing change history to detect the kinds of non-essential differences mentioned above is far from trivial. An automated detection of non-essential differences requires both a characterization of structural changes occurring within statements and an analysis of their impact on the underlying system. In addition, to avoid reconstructing an entire program snapshot for every committed change, the impact of changes must be determined given only a change set, or group of files that were co-committed by a developer [24].

ICSE11, May 21-28 2011, Waikiki, Honolulu, HI, USA

Non-Essential Changes in Version Histories

David Kawrykow
Martin Robillard

McGill

ICSE 2011 . 03/05/2011