GraphXによるグラフ分析処理の実例と入門

freee 株式会社 土佐鉄平

About me

freee 株式会社 土佐鉄平

- ソフトウェアエンジニア
- 三代目巨匠
- チーフセキュリティアーキテクト

@teppei-tosa

http://teppei.hateblo.jp

略歴

- 某金融機関システム子会社で13年勤務
  - 前半8年は基幹系フロントシステムを担当
  - 後半5年は研究開発部門マネージャを担当
    - ビッグデータ技術等を導入推進

- freee株式会社
  - 記帳系、請求書周りを担当
  - CSIRTも兼務
  - 巨匠制度でデータ分析を活用したビジネスプラットフォーム関連の開発を実施

本日お話ししたいこと

グラフ分析ではどういうことができるのか

freeeではグラフ分析をどのように活用しているのか

GraphX でグラフ分析はどのようにできるのか

Overview

- 会計freee とビジネスプラットフォーム構想
  - 会計freeeとは
  - ビジネスプラットフォーム構想とは
  - スマート請求書とは
  - 取引ネットワーク推測機能とは

- 取引ネットワーク推測のにおけるグラフ分析
  - Connected Component によってどれだけつながっているか確認
  - Degreesを確認してつながりすぎているのを見つける

- GraphXによるグラフ分析
  - graphの作り方
  - Connected Component の使い方
  - Pregelとは
  - Connected Component で知る Pregel の使い方

会計freeeとビジネスプラットフォーム構想

freeeのプロダクト

会社設立 freee

(2015年6月リリース)

☆ はじめる

クラウド会計ソフト freee

(2013年3月リリース)

↻ 運営する

クラウド給与計算ソフト freee

(2014年5月リリース)

↗ 育てる

マイナンバー管理 freee

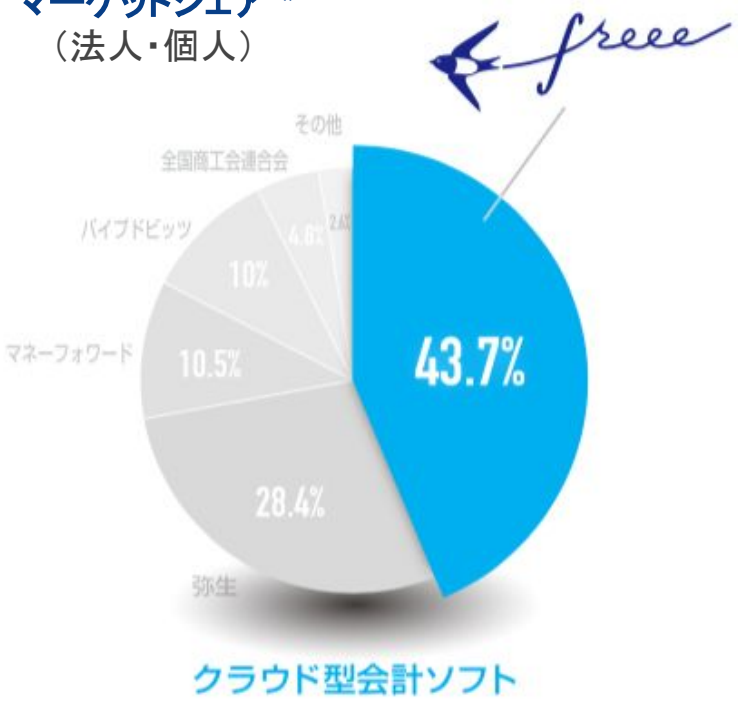
(2015年9月リリース)

◎ 守る

クラウド会計ソフトシェアNo.1

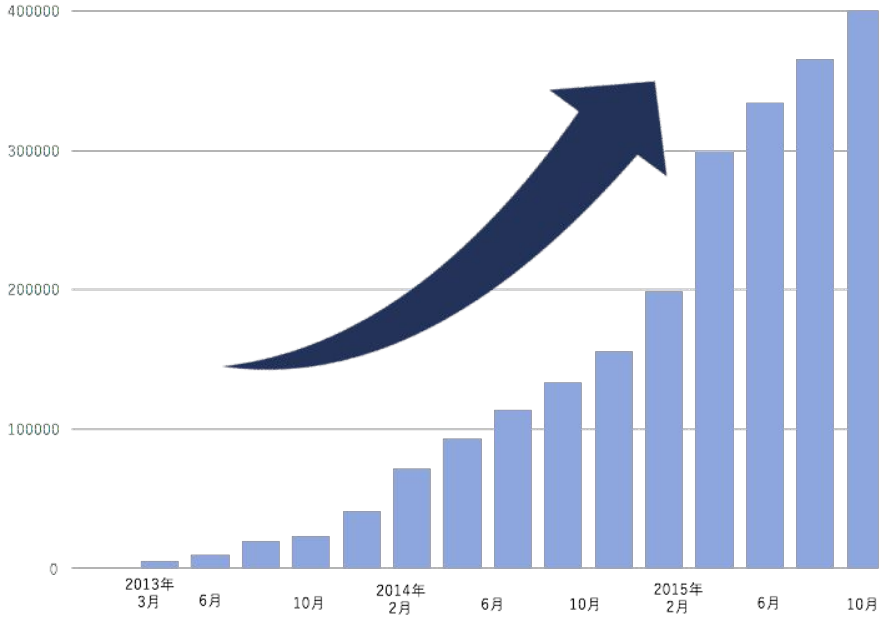
有効事業所数は **70万事業所**を突破
【1年間で2倍以上の伸び】

クラウド会計ソフト市場
マーケットシェア *
（法人・個人）



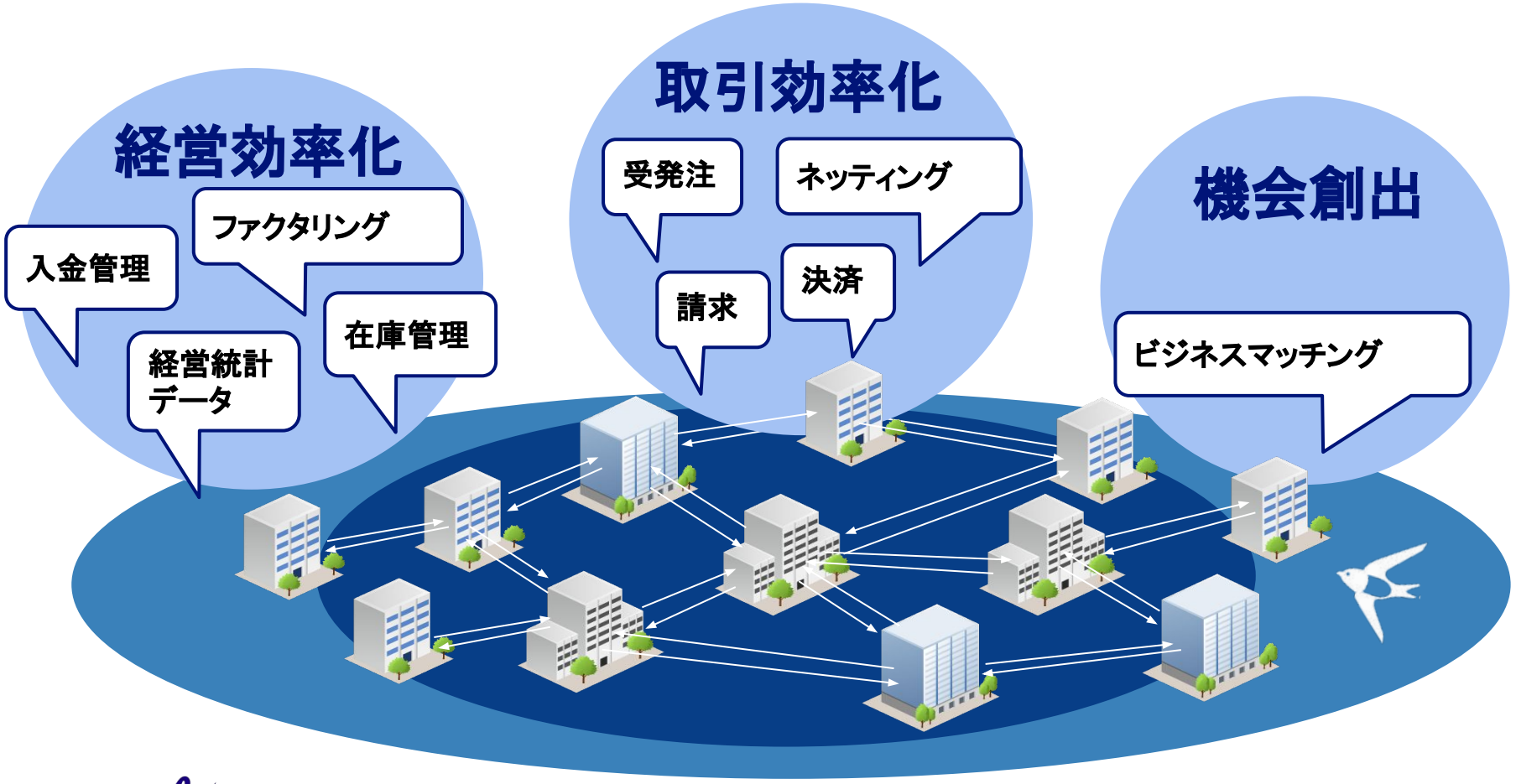
freee 有効事業所数（2016年1月）

会計ソフトおよび給与計算ソフトの利用を目的として登録した事業所のみをカウント



* デジタルインファクト調べ(2015年12月時点)

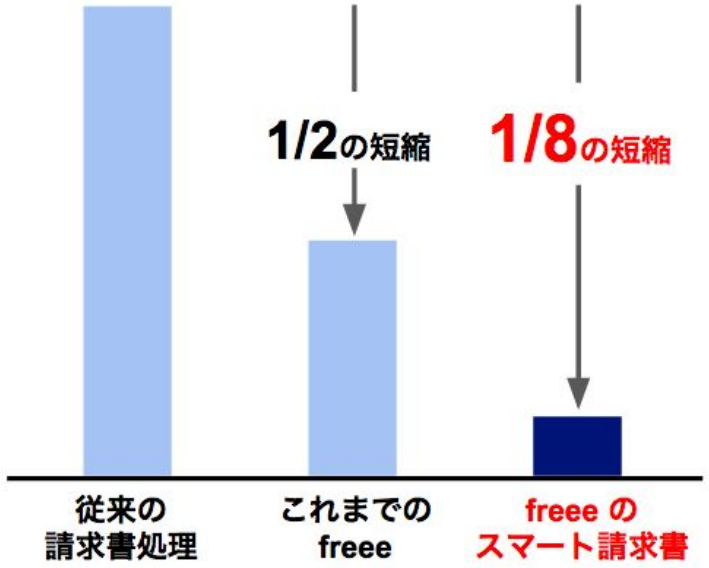
ビジネスプラットフォーム構想とは



スマートビジネスプラットフォーム

ビジネスのやりとりが全てfreee 上で完結

スマート請求書とは



freee 上で請求書のやり取りを完結させることで、

受け取った**請求書の処理時間を1/8に短縮**する

取引ネットワーク推測とは

- 情報公開を許可している企業同士が、よりfreeeを活用できる仕組み
- スマート請求書に続く「企業間取引プラットフォーム」施策の第2弾

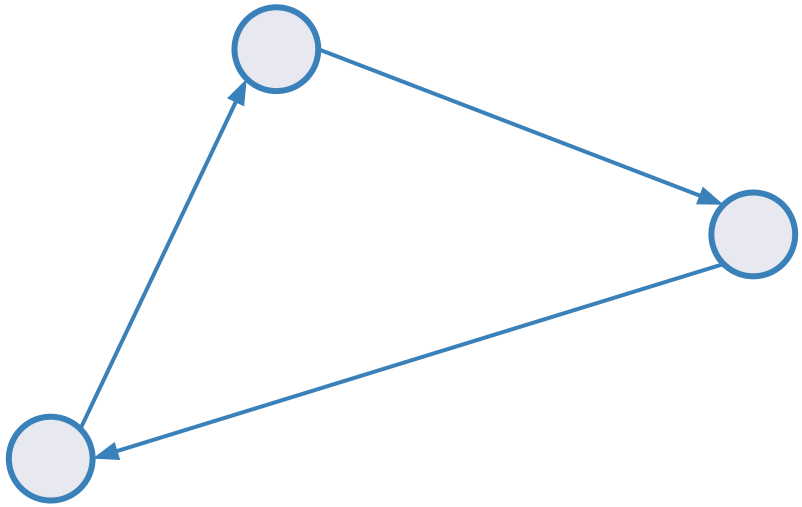
取引ネットワーク推測とは

- 事業所に設定されている「取引先」と、他の事業所の一致予測を実施
- freeeの中でも、実取引としてもつながっている事業所を見つけ出す
- 事業所の「公開設定」機能を提供
  - 予測確度に応じて、公開対象と公開範囲を設定可能
  - 例：確度・高に対しては、住所を公開
- 取引先が freee の利用事業所で、情報公開しているならば、スマート請求書の利用を推奨したり、相手事業所の名称や住所をコピー可能

取引ネットワーク推測におけるグラフ分析

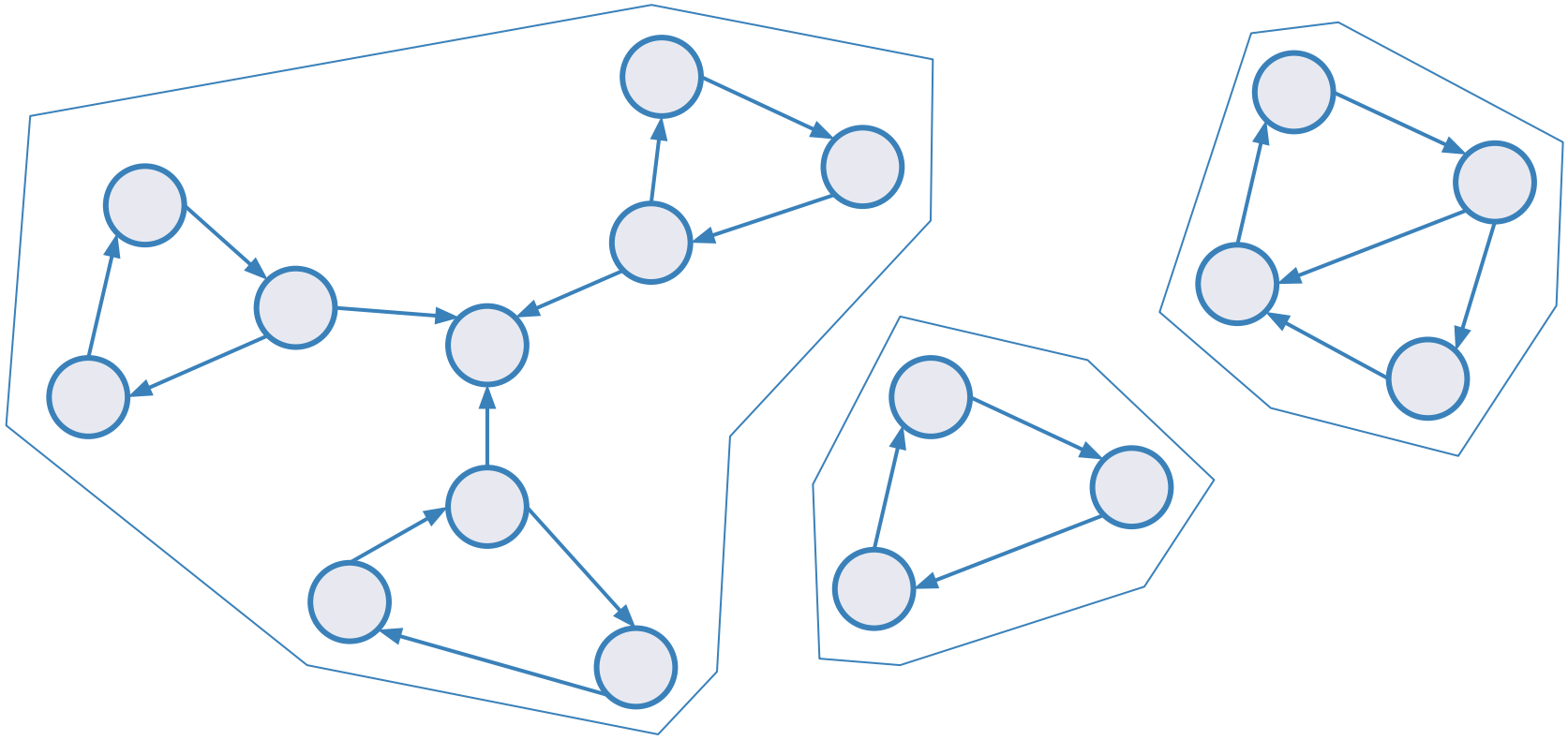
Graph とは

- 頂点（vertex）と辺（edge）の集まり
- 辺は方向（direction）を持つことができる
- 頂点と辺はそれぞれ属性（attribute）を持つことができる



Connected Component による分析

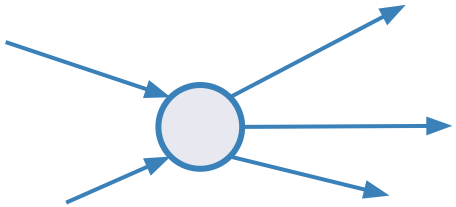
- Connected Component（連結成分）とは、（難しい定義はよくわからないけど）、要は、辺でつながっている頂点の集まり



- 大きすぎる 連結成分 が存在するようであれば、不正に辺を集めている頂点が存在することを疑う
- ただしシェアが拡大すれば、ひとつの連結成分がほとんどの頂点を含むようになる

Degrees の分析

- 頂点とつながる辺の数を「次数（degrees）」と呼ぶ
- 頂点に入ってくる辺の数を「入次数（in-degreees）」
- 頂点から出る辺の数を「出次数（out-degrees）」



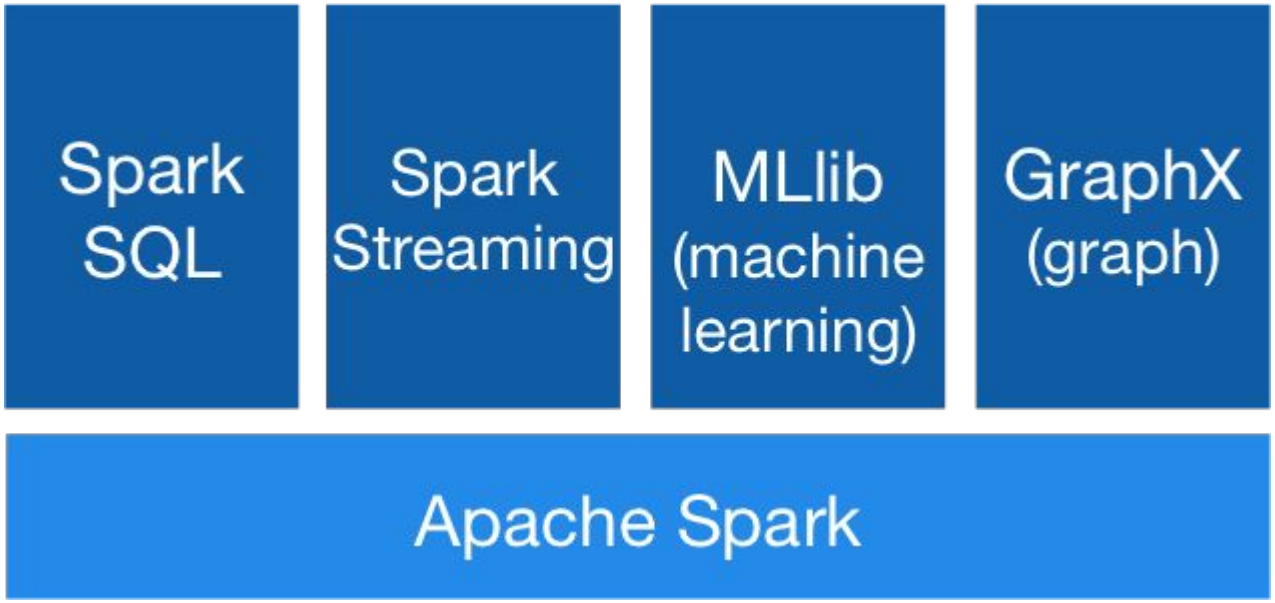
```
degrees: 5, in-degrees: 2, out-degrees: 3
```

- 次数の多い順に sort して 上位10件を確認するなどして、巨大な連結成分の原因となっている頂点を探す
- そうすると、紐つけロジックの不備が大体推測できて、見直しをする

GraphXによるグラフ分析

GraphXとは

- GraphX は、Apache Sparkのコンポーネントの一つで、大容量グラフ構造データの並列分散処理を実装するためのフレームワーク
- ETL、機会学習、クエリ、グラフ分析をひとつのアプリケーション内でシームレスに実装できる
- GraphX自体が Spark で実装されているため、GraphX の内部実装の把握と、Sparkアプリを実装することの距離が近いことも魅力



graph の作り方

- 例えば別の大規模グラフデータ分析フレームワークの Apache Giraph では、以下のようにグラフデータの形に変換したデータを読み込む必要がある

```
[1, 100, [[2, 100]] ]
[2, 200, [[3, 200]] ]
[3, 300, [[1, 100]] ]
```

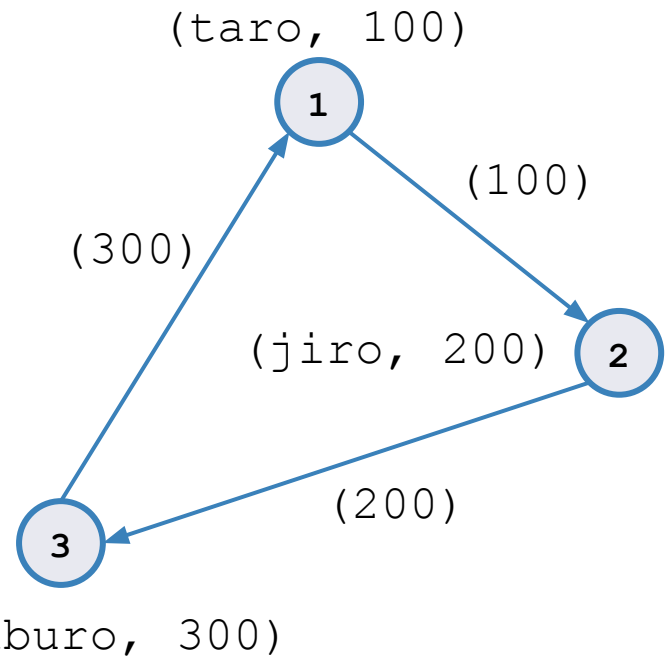
- GraphXでは、以下のように表構造のデータから
  グラフオブジェクトを作ることができる

master.csv

id	name	value
1	taro	100
2	jiro	200
3	saburo	300

connection.csv

id	from	to	value
1	1	2	100
2	2	3	200
3	3	1	300

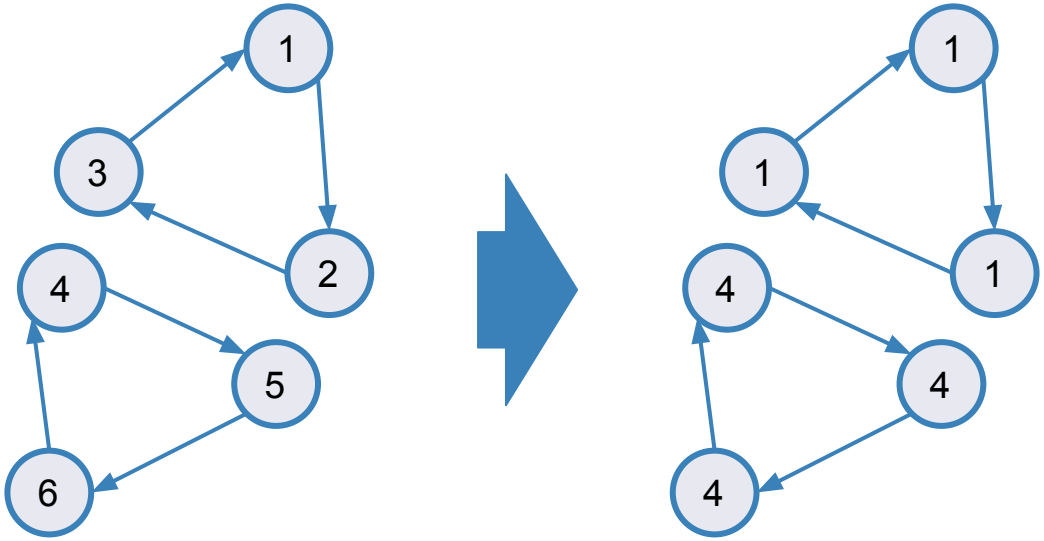


graph の作り方

```
// 元データを読み込み
val master = sc.textFile("master.csv")
val conn = sc.textFile("connection.csv")

// 頂点データを生成
val vertices:RDD[(VertexId, String)] = master.map( line => {
    val cols = line.split(",")
    (cols(0).toLong, cols(1))
})

// 辺データを生成
val edges:RDD[Edge[Long]] = conn.map( line => {
    val cols = line.split(",")
    Edge(cols(1).toLong, cols(2).toLong, cols(3).toLong)
})

// 頂点データと辺データからグラフを生成
val graph:Graph[String, Long] = Graph(vertices, edges)
```

Connected Component の分類の仕方

- 各 vertex の id をそれぞれの value にセットして、各 connected component の vertex の value をその中で最小の値の id になるように分類される



- 実装はこれだけ

```
val cc:Graph[Long, Int] = graph.connectedComponents
```

degrees の取得

- 次数の取得

```
val degrees:VertexRDD[Int] = graph.degrees
```

- 出次数の取得

```
val outDegrees:VertexRDD[Int] = graph.outDegrees
```

- 入次数の取得

```
val inDegrees:VertexRDD[Int] = graph.inDegrees
```

- 結果は id と 次数の tuple で得られる

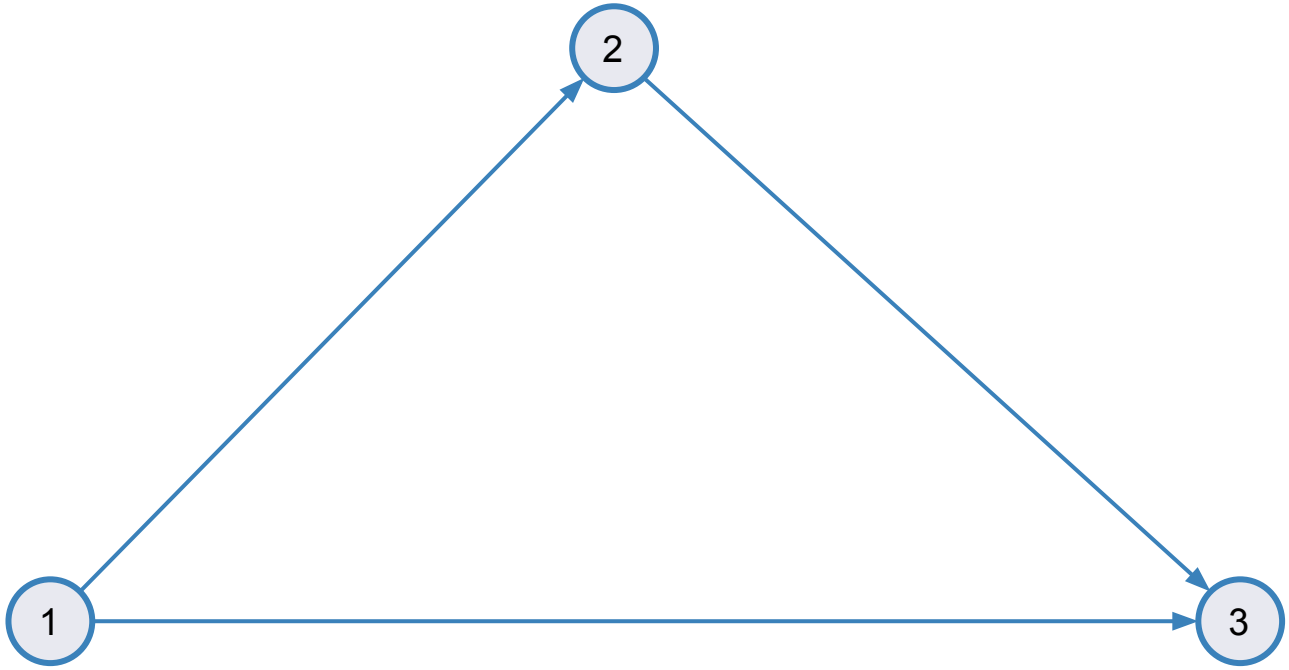
Pregel とは

- Pregel は Google が開発したグラフアルゴリズムを実装する ためのスケーラブルなプラットフォーム実装
- Google の論文を元に GraphX 内で関数が実装されて利用可能
- API

```
Pregel(
 graph,   // 処理対象□グラフデータ
 initialMsg:A,   // 最初□のイテレーションで送信するメッセージ
 maxIter:Int,   // 最大何回イテレーションするか
 activeDir:EdgeDirection   // どちら方向にメッセージ送信するか
)(
 // イテレーション毎に頂点で稼動させる処理（A□：受信したメッセージ）
 vprog:(VertexId, VD, A) => VD,
 // イテレーション毎に辺上に何を流すか決定する処理
 sendMsg:(EdgeTriplet[VD, ED]) => Iterator[(VertexId, A)],
 // 複数 □Edge からメッセージを受信した時□処理
 mergeMsg:(A, A) => A
 // 処理結果として新しいGraphデータを返す
):Graph[VD, ED]
```

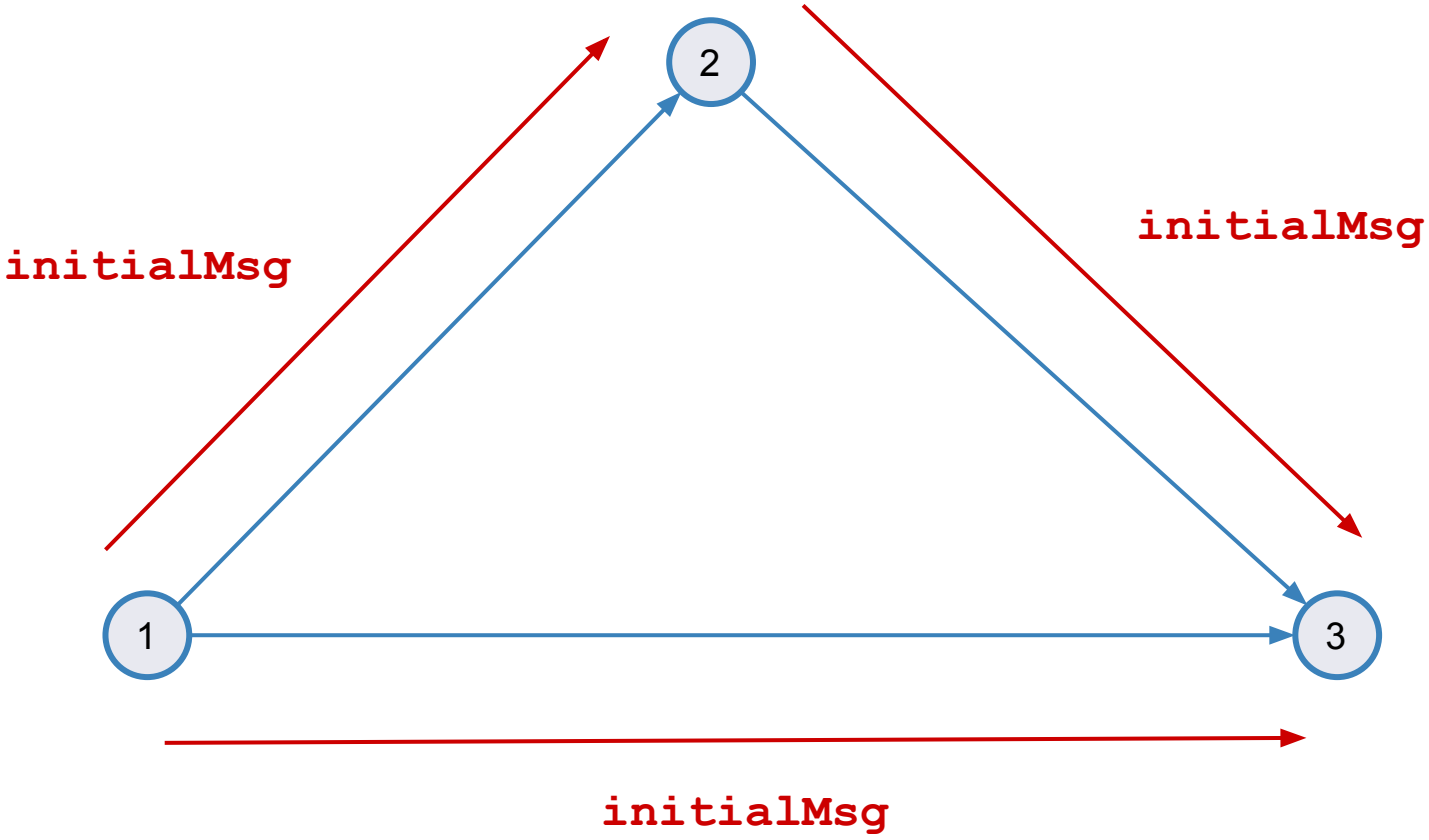
Pregel とは

- 簡単なグラフを例に考えてみる



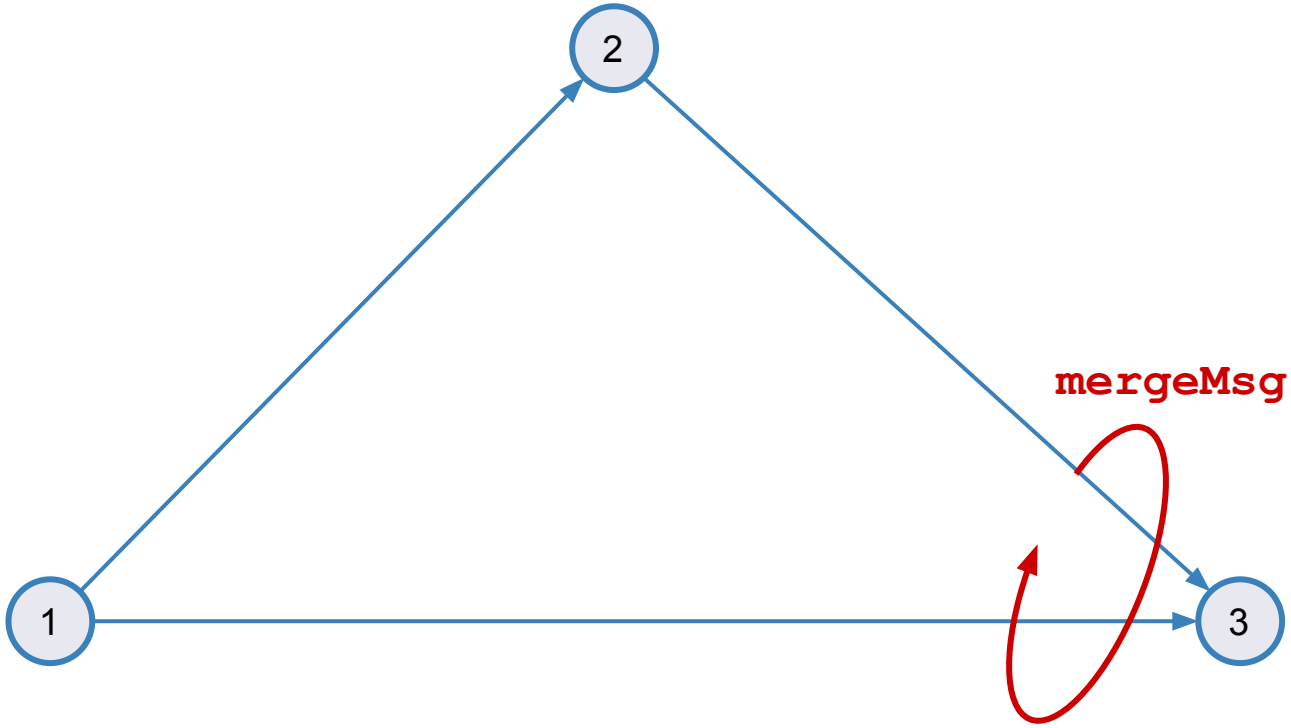
Pregel とは

- 引数 initialMsg が全ての辺で流れる
- この処理は最初の iteration でだけ稼働する



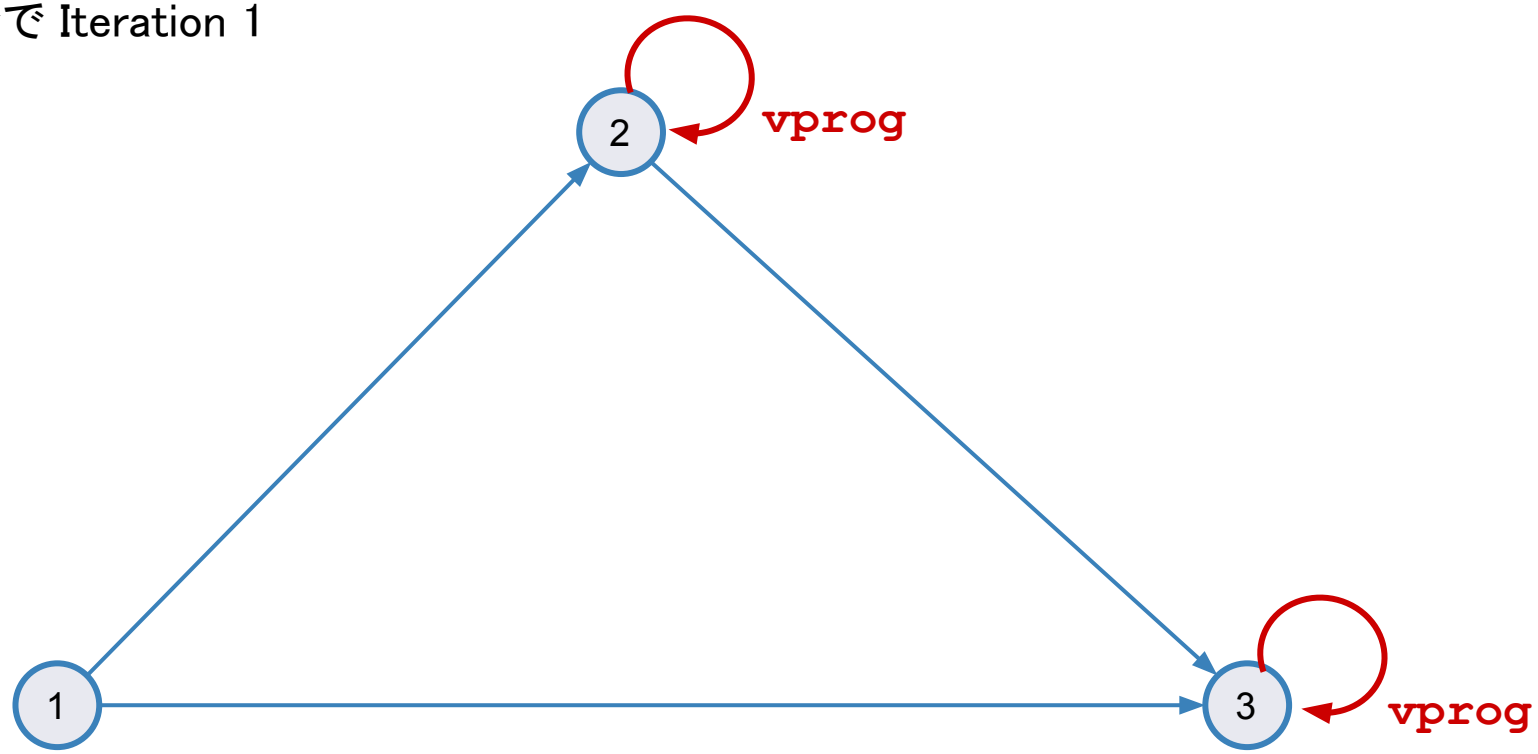
Pregel とは

- 複数の辺が入ってきている頂点で、各辺から渡ってきた値をどう merge するか実装した mergeMsg 関数が稼働する



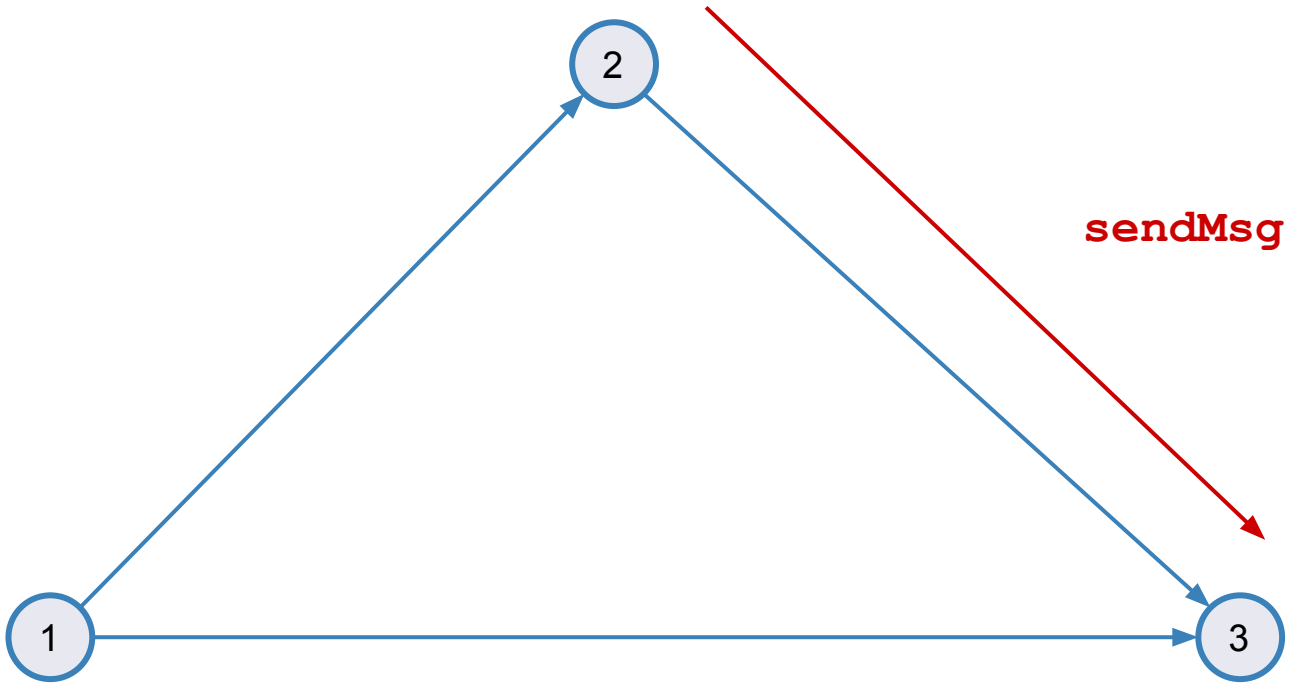
Pregel とは

- 辺が入ってきている全ての頂点で、入ってきた値をどう処理するかを実装したvprog が稼働
- ここまでで Iteration 1



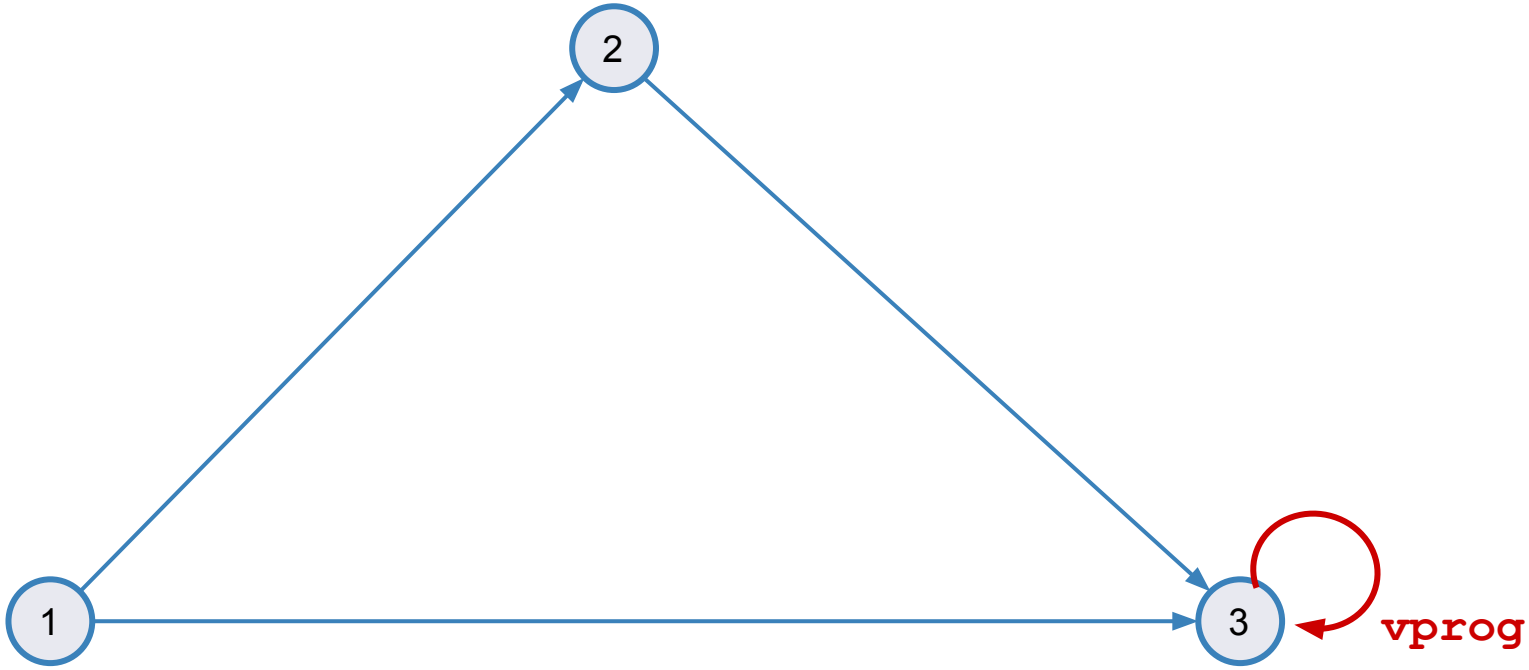
Pregel とは

- 前の iteration で値を受信した頂点から出ている辺で、どんな値を送るか実装した sendMsg が稼働



Pregel とは

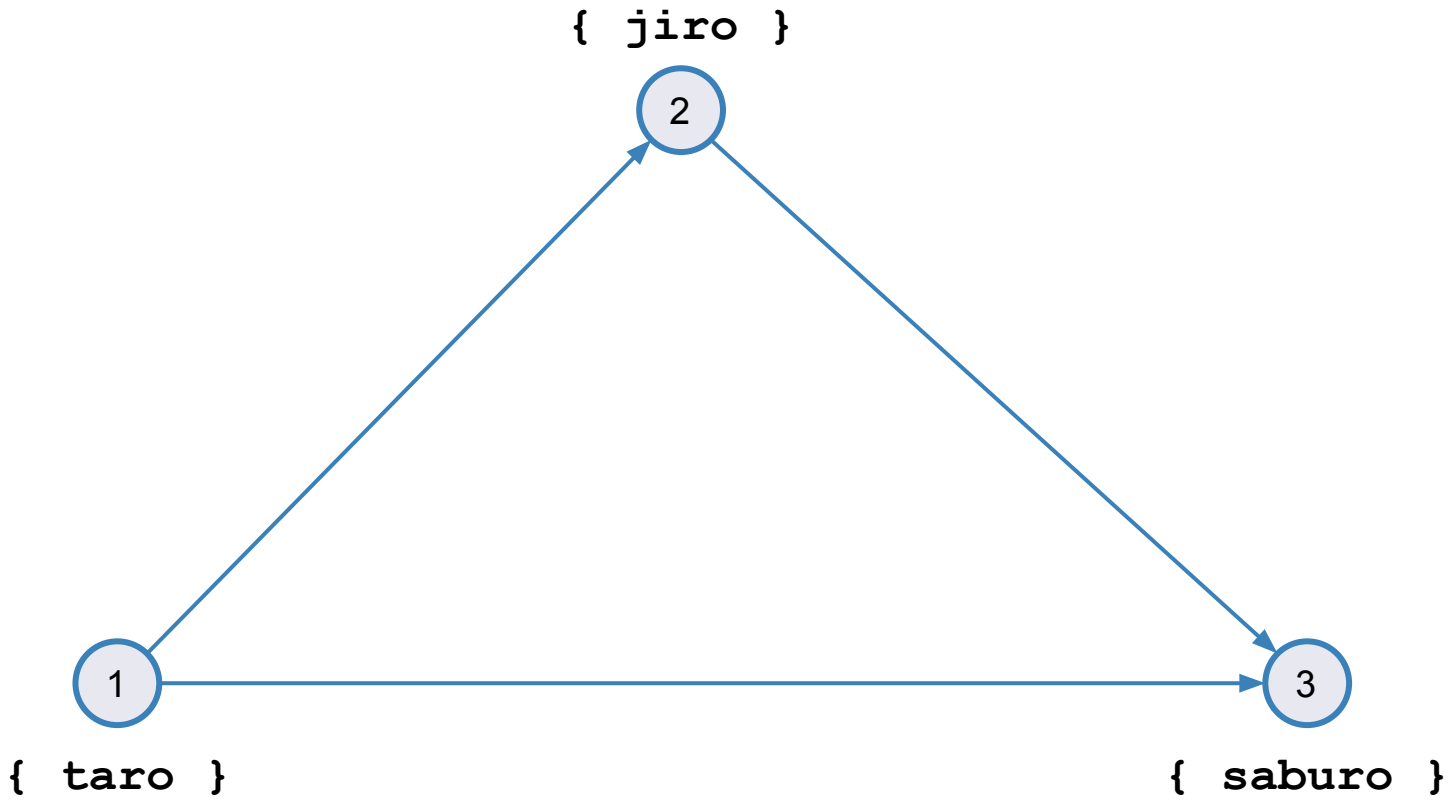
- 辺が入ってきている全ての頂点で、入ってきた値をどう処理するかを実装したvprog が稼働
- 複数値が入ってくる頂点があればこの前に mergeMsg が稼働



- 以上の Iteration が以下のどちらかの条件を満たすまで続く
  - 引数 maxIter で指定された回数続く
  - どの頂点からもメッセージが送信されなくなる

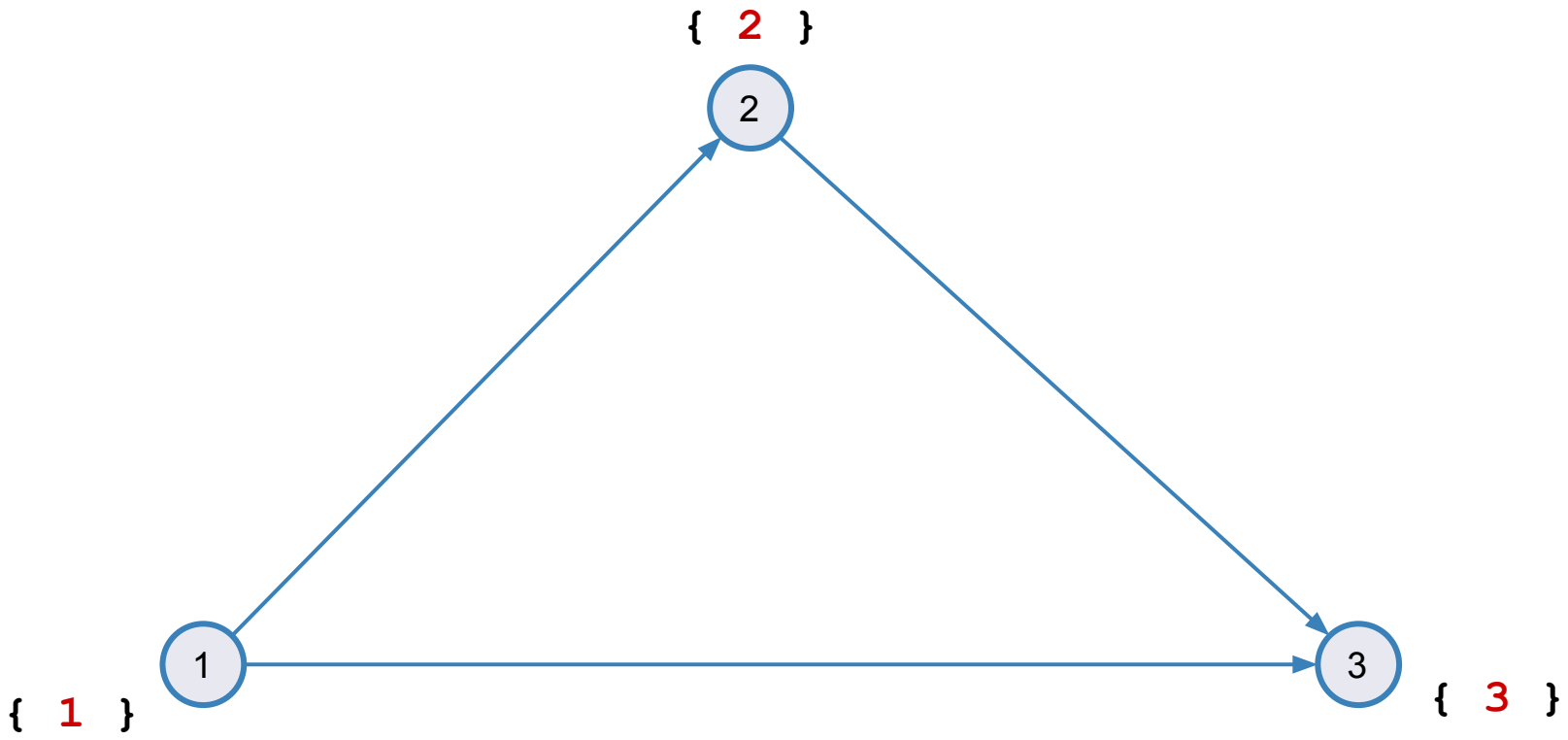
Connected Conponent は Pregel をどう使っているか

- 簡単なグラフを例に考えてみる



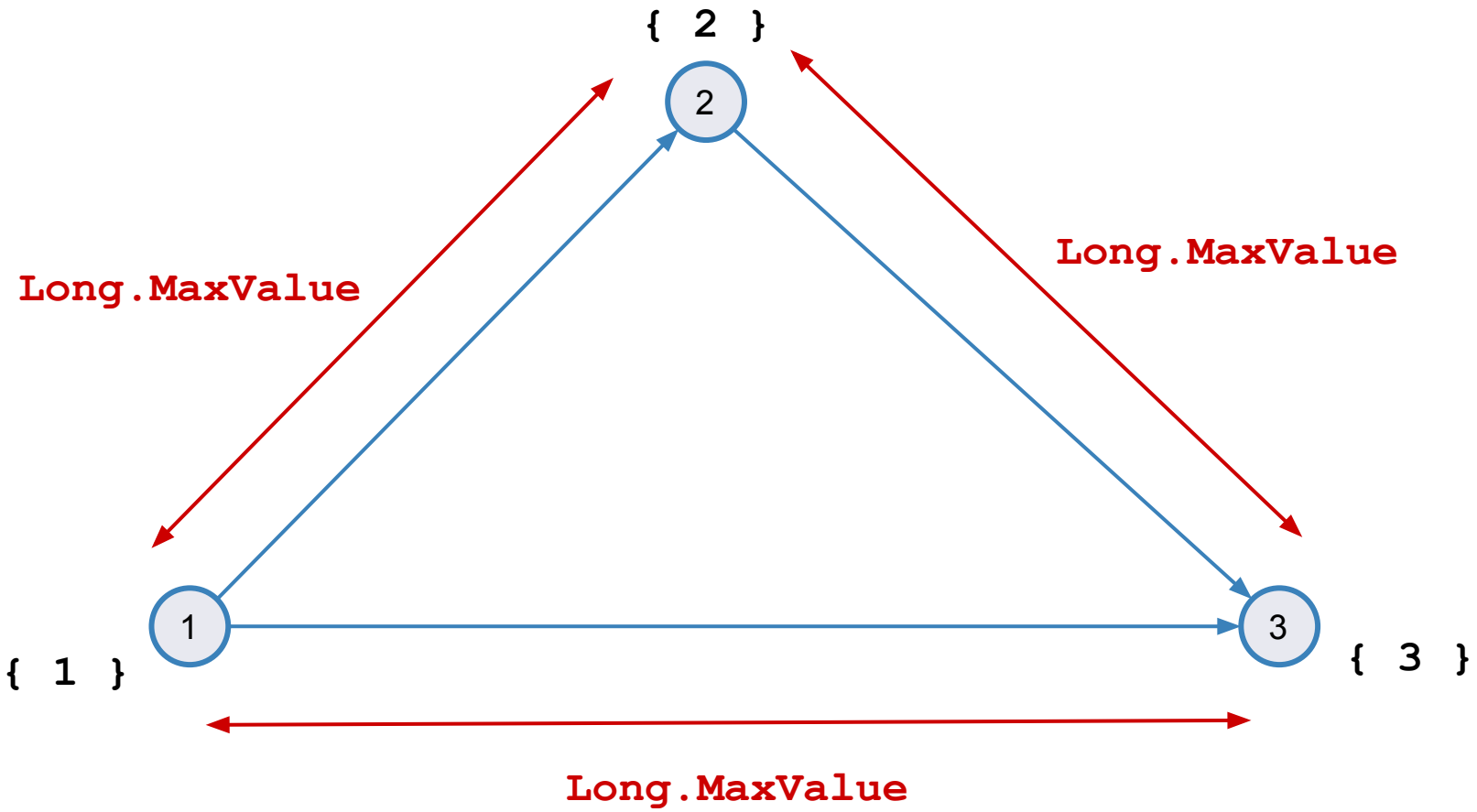
Connected Conponent は Pregel をどう使っているか

- 各Vertexの値をVertexのidに置き換える



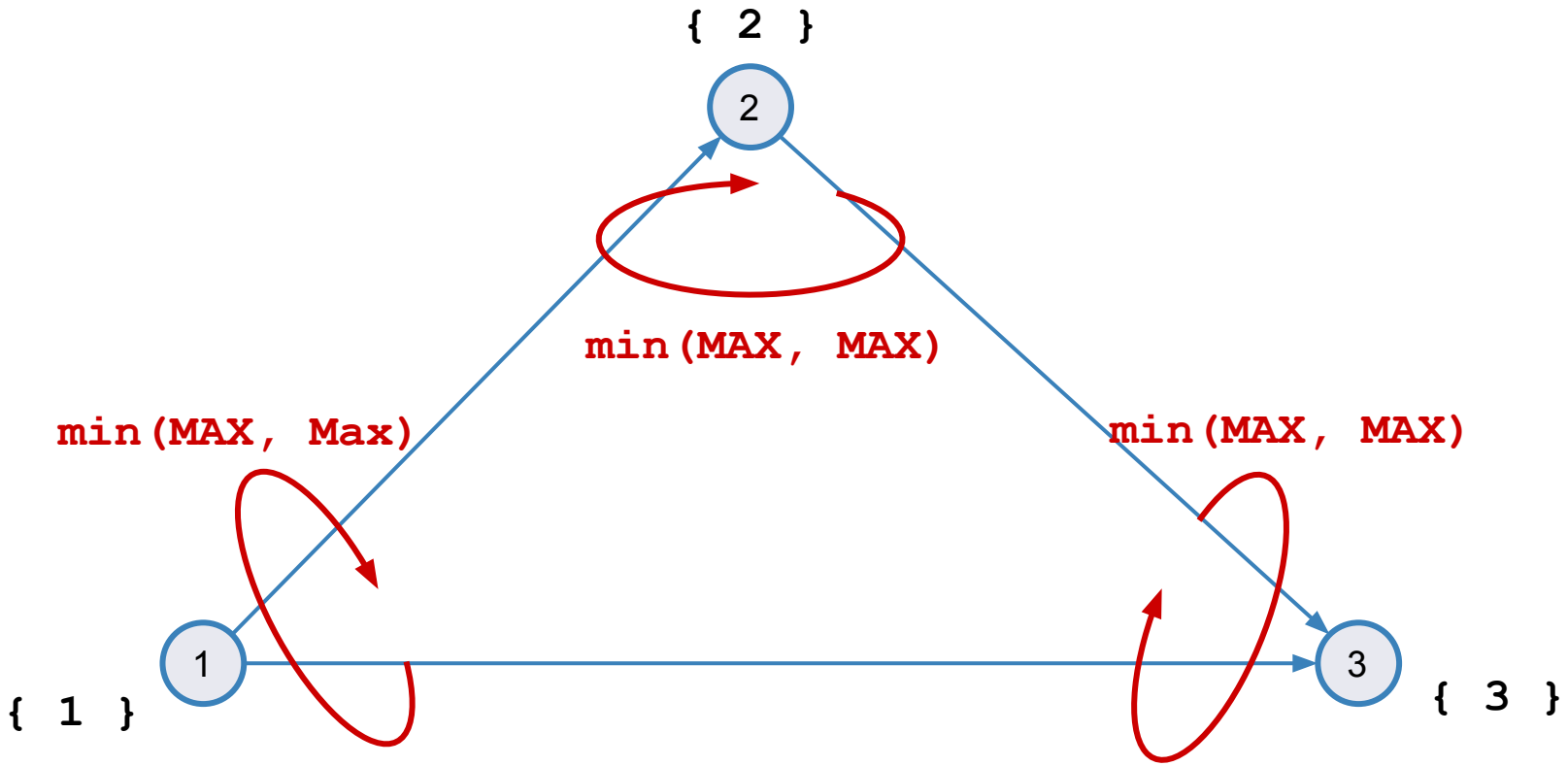
Connected Conponent は Pregel をどう使っているか

- initalMsg として Long.MaxValue を全Edgeで双方向に送信する



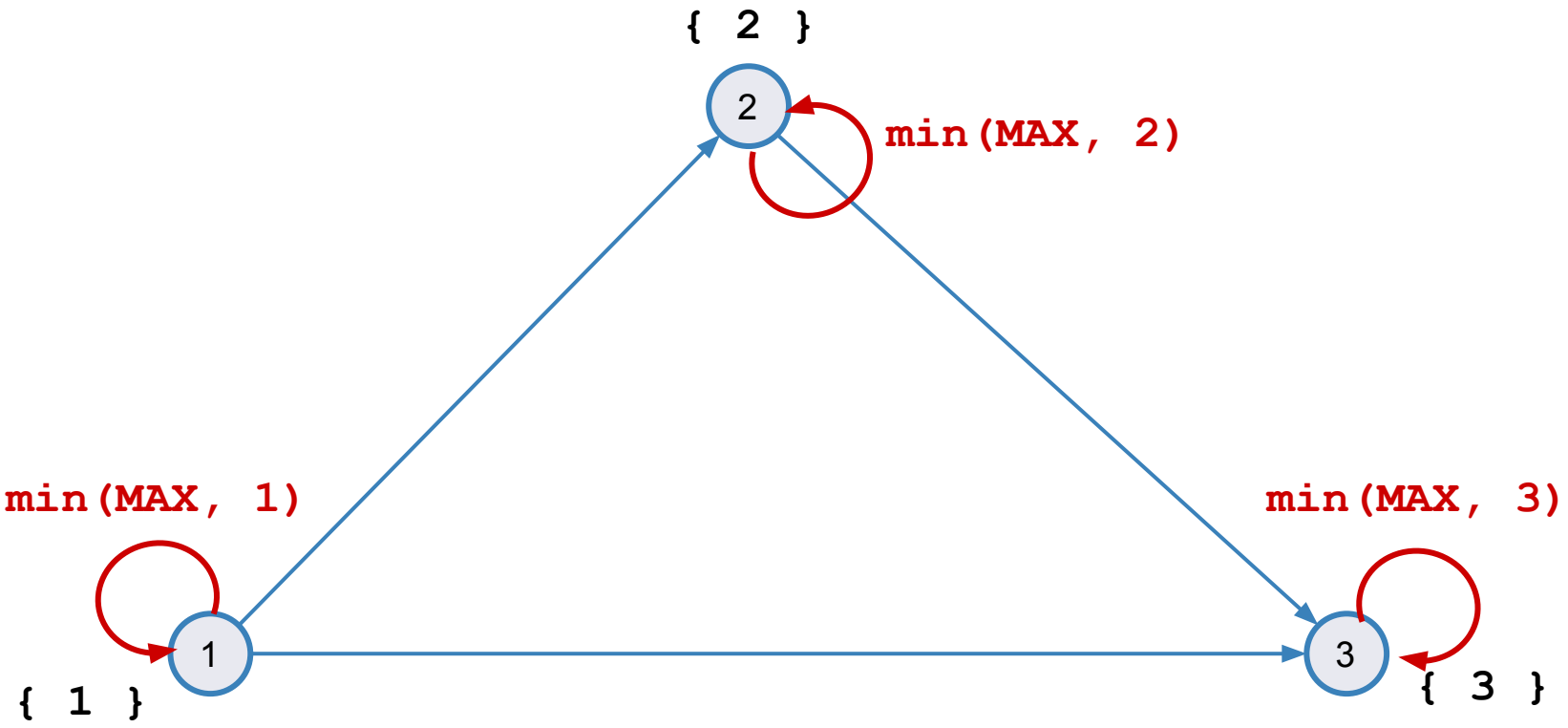
Connected Conponent は Pregel をどう使っているか

- 複数Edgeが集まっているVertexは流れてくる値の小さい方をとる（つまりLong.MaxValue）



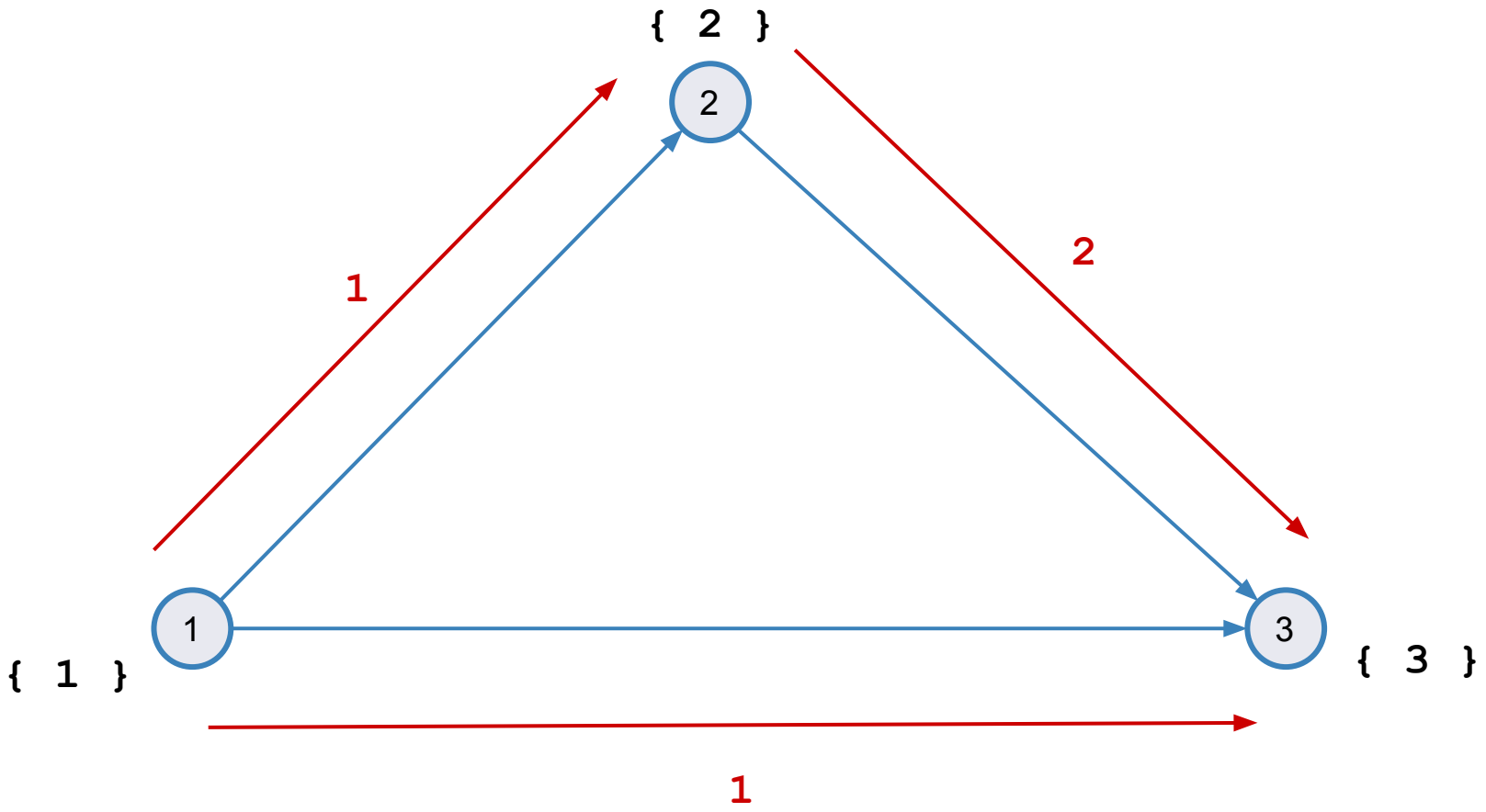
Connected Conponent は Pregel をどう使っているか

- 受信したmsgとVertexが持っている値の小さい方をとる（つまりVertexの値の更新なし）



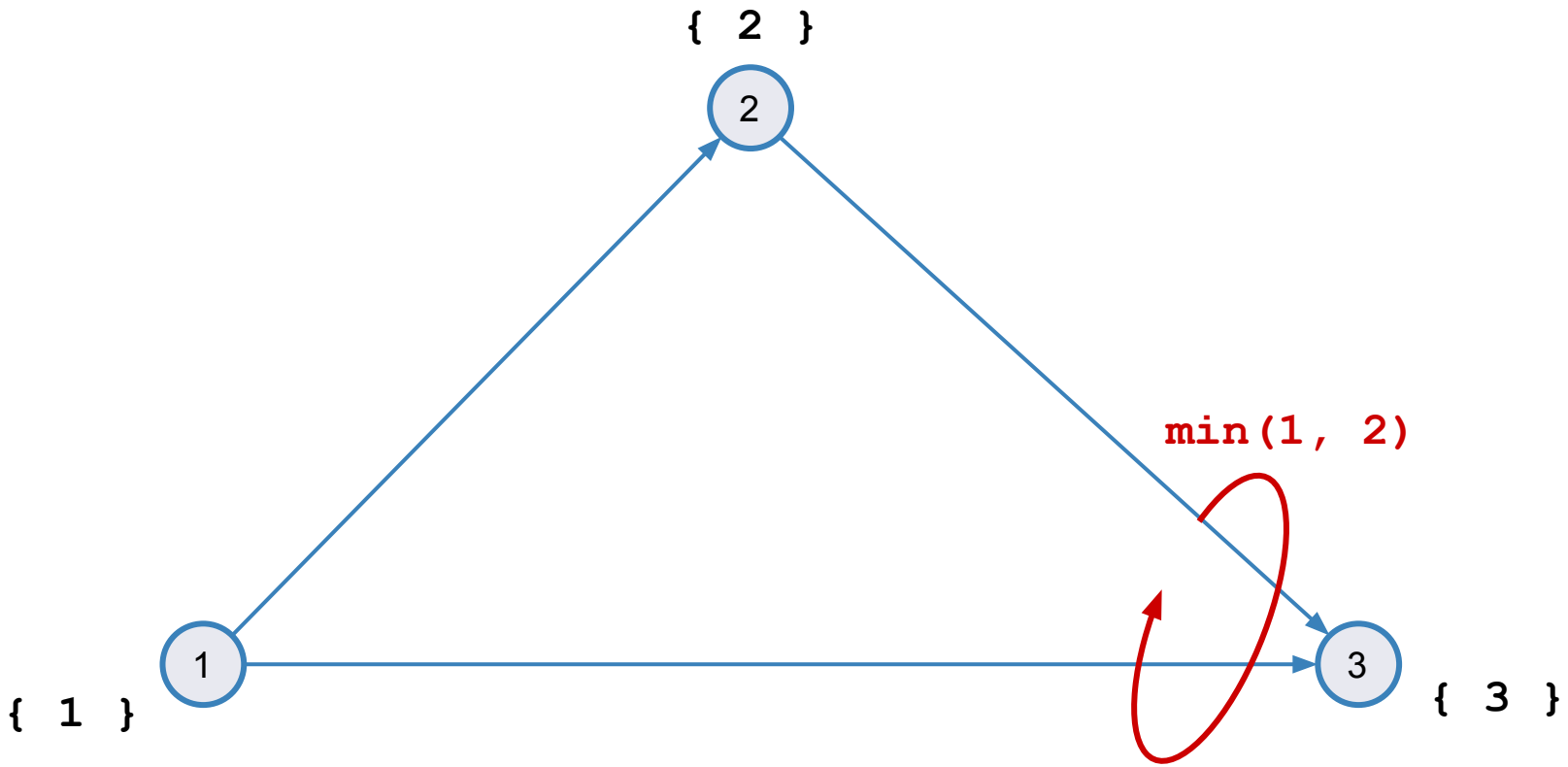
Connected Conponent は Pregel をどう使っているか

- msgを受信したvertex（つまり全てのvertex）から、小さい方の値を反対側に流すというロジックに従ってmsg送信



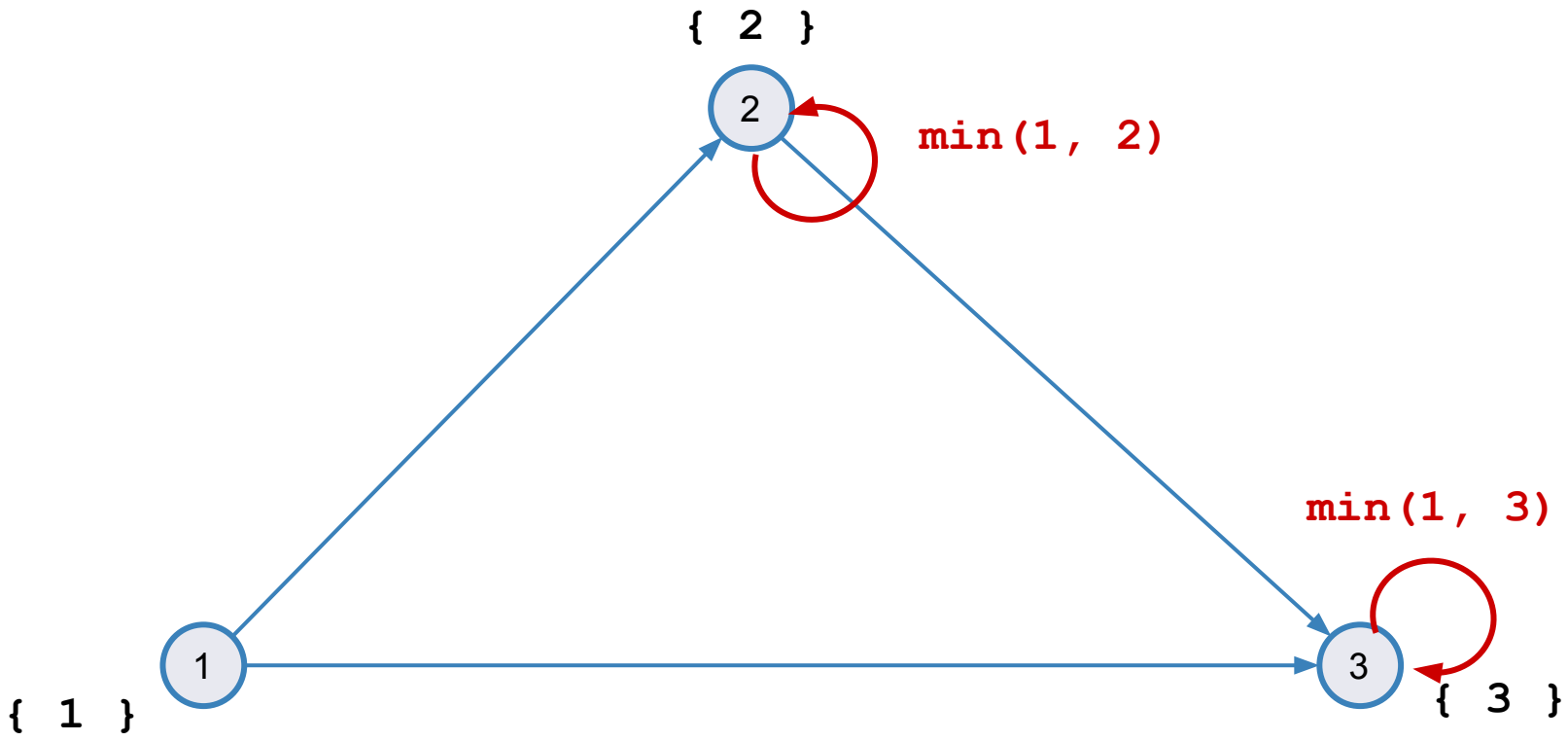
Connected Conponent は Pregel をどう使っているか

- 複数Edgeが集まっているVertexは流れてくる値の小さい方をとる



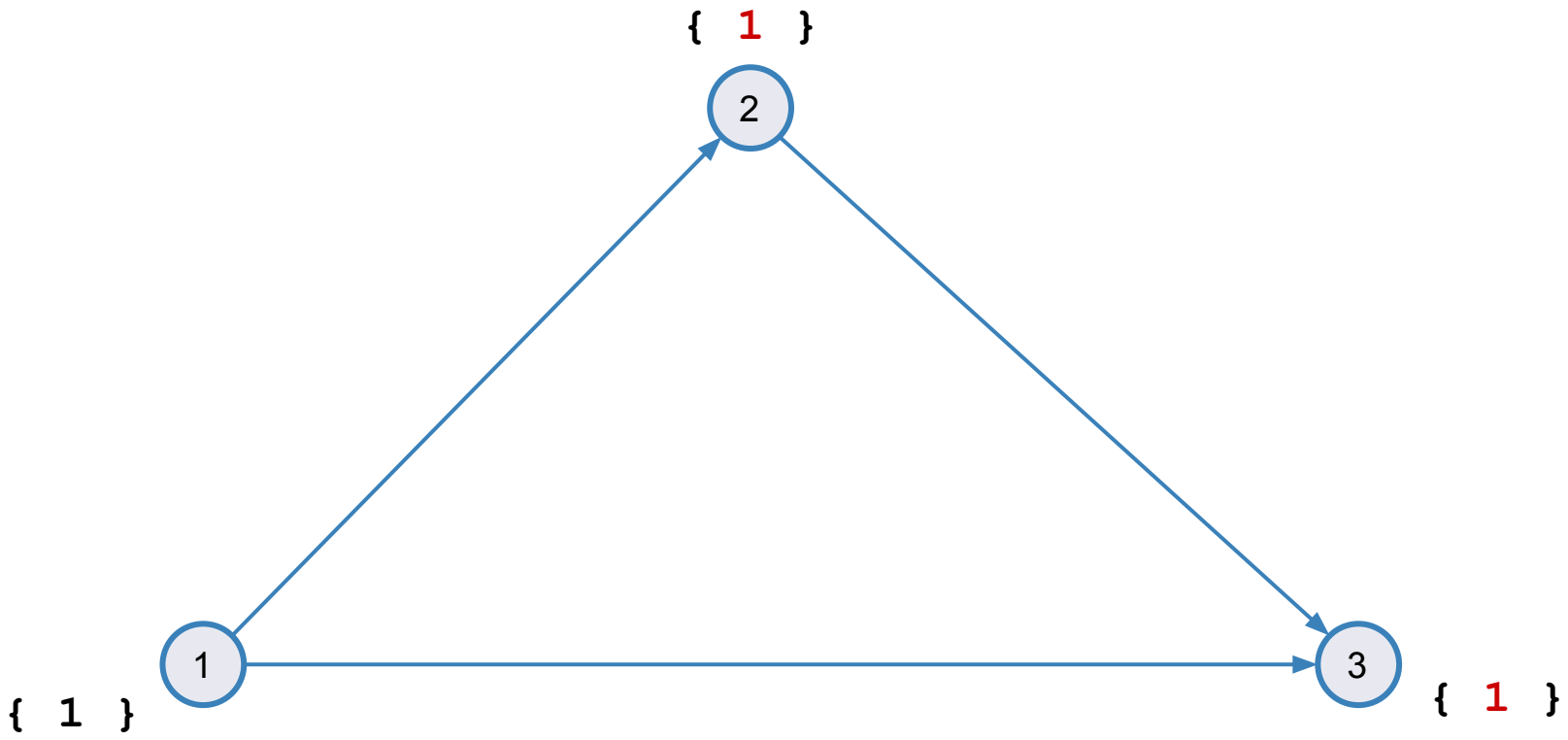
Connected Conponent は Pregel をどう使っているか

- 受信したmsgとVertexが持っている値の小さい方をとる（つまりVertexの値の更新なし）



Connected Conponent は Pregel をどう使っているか

- 受信したvertexの値が更新される



- 受信したvertexから値送信しようとするが、どの辺もsrcもdstも同じ値なので送信がされず、これで Iteration 終了

Connected Conponent は Pregel をどう使っているか

- 実装(1)

```
val cc = Pregel(
  // 各頂点の値をVertexのIDに変換したグラフをインプットにする
  graph.mapVertices { case (vid, _) => vid },
  // Iterationの最初に大きな値をEdgeに流す
  Long.MaxValue,
  // Iterationの最大回数を変数で指定できるようになっている
  maxIterations,
  // msgはEdge上を双方向に流す
  EdgeDirection.Either
)(
  // Edgeを流れてきた値とVertexで持っている値の小さい方をとる
  vprog = (id, attr, msg) => math.min(attr, msg),
  // Edge上をどうmsg流すかは別途関数を定義
  sendMsg = sendMessage,
  // 同じVertexに複数のEdgeからmsg来たら、小さい方をとる
  mergeMsg = (a, b) => math.min(a, b)
)
```

Connected Conponent は Pregel をどう使っているか

- 実装(2)

```
// Edge上をどうmsg流すか別途定義した関数
def sendMessage(edge: EdgeTriplet[VertexId, ED])
  : Iterator[(VertexId, VertexId)] = {
  // src側の値が小さければsrcの値をdstに流す
  if (edge.srcAttr < edge.dstAttr) {
    Iterator((edge.dstId, edge.srcAttr))
  // dst側の値が小さければdstの値をsrcに流す
  } else if (edge.srcAttr > edge.dstAttr) {
    Iterator((edge.srcId, edge.dstAttr))
  // srcとdstの値が同じなら何も流さない
  } else {
    Iterator.empty
  }
}
```

まとめ

まとめ

データを「表構造」で扱っていてはできない分析が
グラフ分析ならできる

GraphX なら大規模なグラフデータ分析が
ETL処理や機会学習処理とシームレスに実装できる

GraphXやろう！

おまけ:GraphX から GraphFrame へ

- Spark のベースI/Fが RDDからDataFrame に以降するのに伴い、GraphX と DataFrame が統合したものが、GraphFrame。
- DataFrame をベースに 大規模グラフ分析 を可能にするフレームワーク
- Neo4jで使われている CypherQuery が利用できるのが特徴
- まだ ver 0.3 で、現状は GraphX から手を出しても大丈夫
- いずれは GraphFrame に移行

参考サイト

- GraphX Programing Guide
- GraphX Advent Calendar 2014
- Spark graph framesとopencypherによる分散グラフ処理の最新動向
- GraphFrame

Question?

スモールビジネスに携わるすべての人が
創造的な活動にフォーカスできるよう