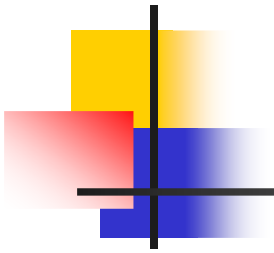
さあその先へ、OpenLDAP パフォーマンスチューニング

2010/02/26

日本LDAPユーザ会

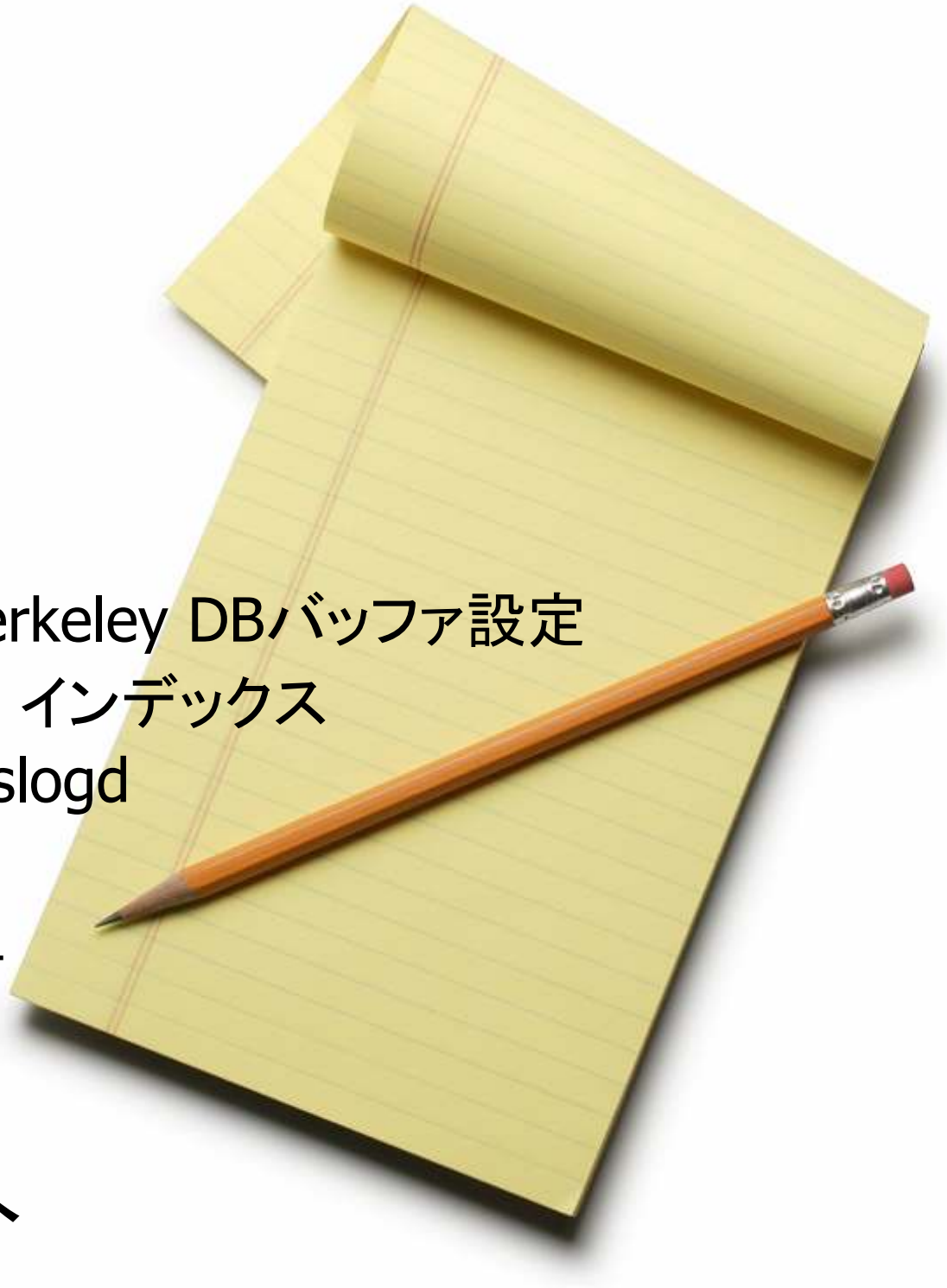


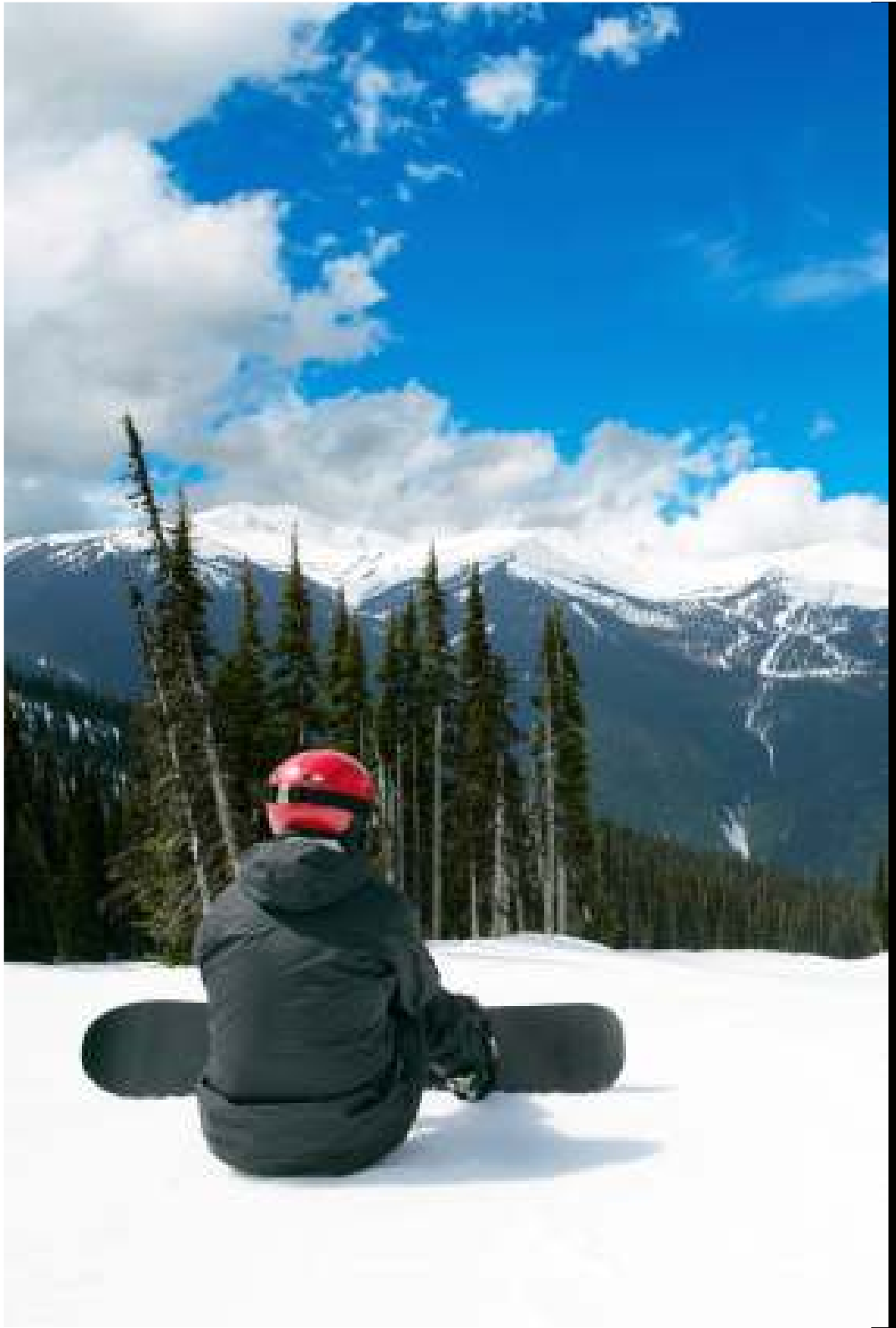
はじめに

- OpenLDAPは、他のオープンソースソフトウェアと同様に、わずかなマシンリソースでも動作させることができるディレクトリサーバです
- OpenLDAPは、わずかなチューニングを施すことで、飛躍的に性能を向上させられる可能性を秘めたソフトウェアです
- 本セッションでは、OpenLDAPサーバのチューニングの勘所をご紹介します

目次

- 試験環境
- チューニング
  - DB_CONFIGと、Berkeley DBバッファ設定
  - 検索処理フローと、インデックス
  - ログファイルと、syslogd
  - エントリキャッシュ
  - ID リストキャッシュ
  - スレッド数
- もう一歩、その先へ





試験環境

試験環境

サーバ機

CPU:2.53GHz Quad Core×2
メモリ:12G
ディスク:146GB SAS 10rpm×4
ディスクコントローラ:512Mキャッシュ

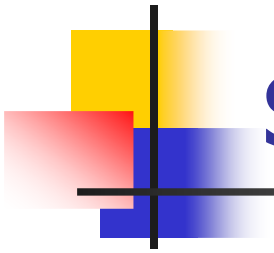
負荷サーバ

OS:CentOS 5.4 x86_64
JVM:Sun 64bit 1.6.0_18
検証ツール: slamd-2.0.0

OpenLDAPサーバ

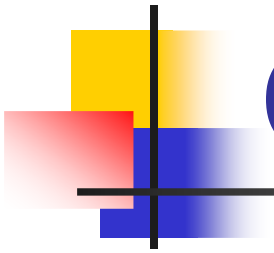
OS:CentOS 5.4 x86_64
Berkeley DB:4.7.25
OpenLDAP: 2.4.19

※OpenLDAP 2.4.19 (+ Berkeley DB 4.7.25) は、本資料作成時点での最新の安定版リリースです。



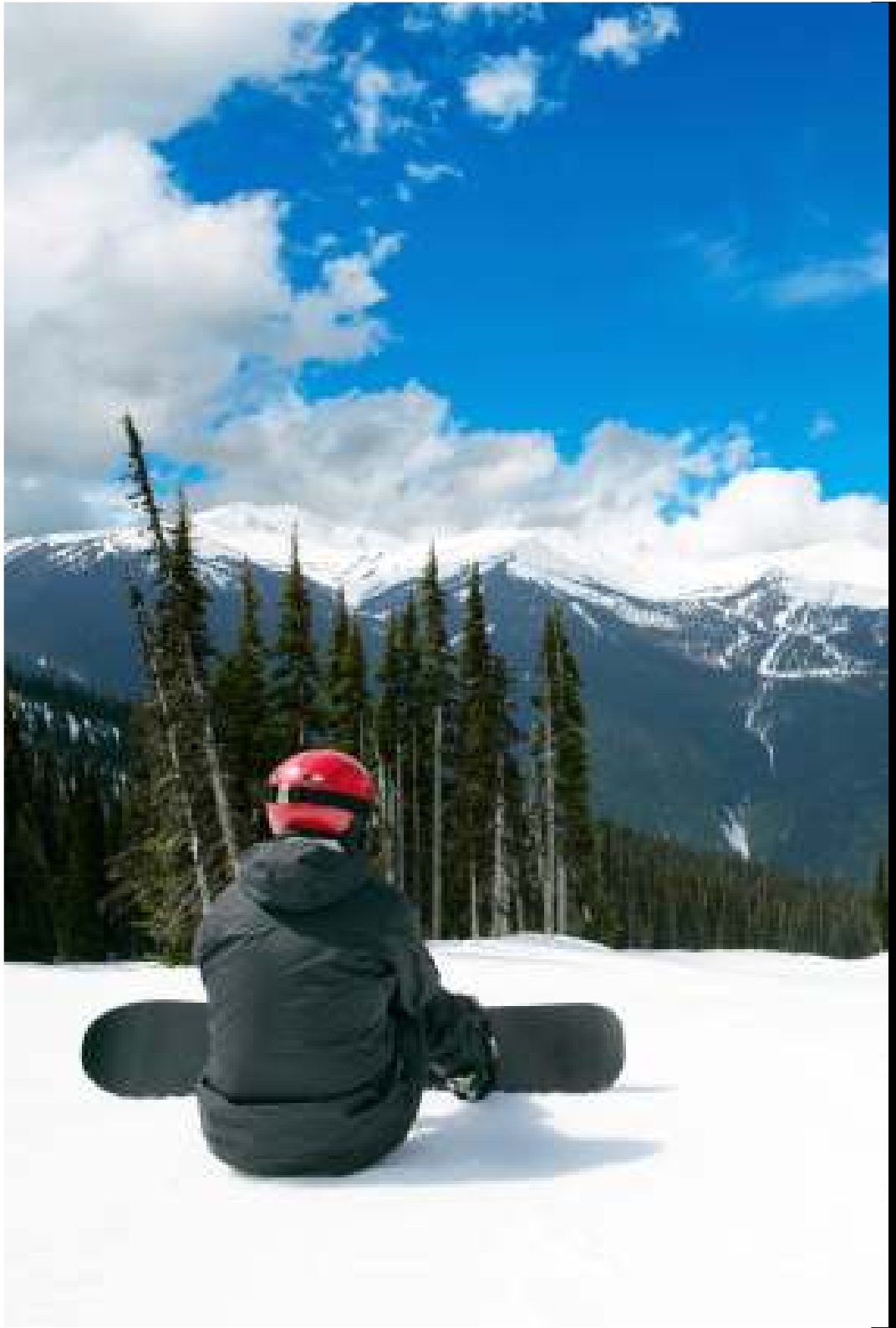
slamdからの負荷内容

- LDAPプロトコルでの様々な試験が可能なツール
- 検証に用いた負荷の内容
  - Bind(認証)処理 １：Search(検索)処理 ３　の割合
    - LDAP Mixed Load
  - 1万のエントリを対象
- 検証時のセッション数
  - 100セッション (10クライアント　×　10スレッド)



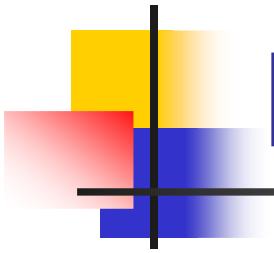
OpenLDAPサーバの初期設定

- 特別なオプションなしでインストール
- 試験中に、ログファイルを参照してエラーがないかを確認できる状態
  - syslogdへstatsログを送付し、記録
- 試験中に、ldapsearchにて、slapdの動作状況を確認できる状態
  - monitorデータベースを利用



DB_CONFIG

を知っておこう！

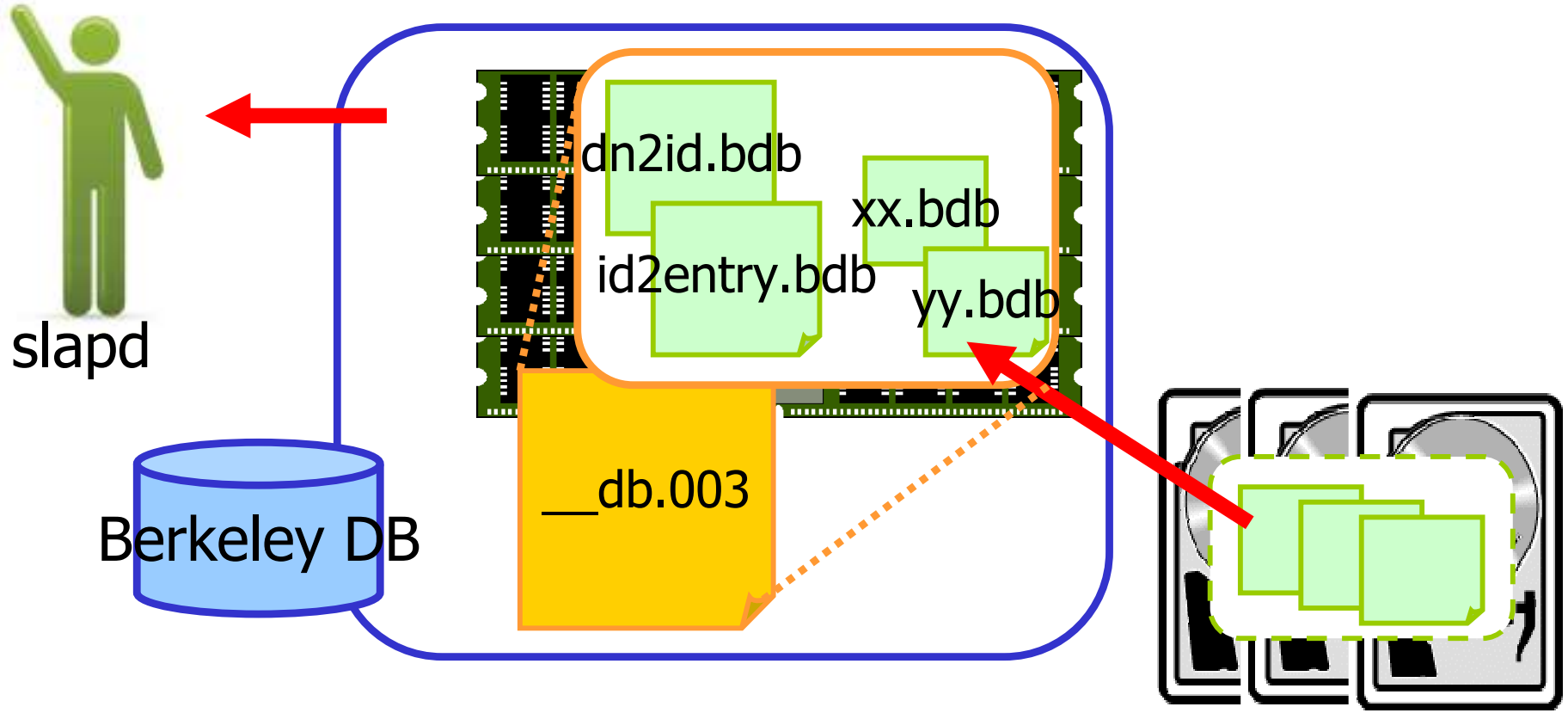


DB_CONFIG利用の意味

- ざっくりBerkeley DBをチューニング
  - Berkeley DBへの参照処理性能の向上
    - set_cachesizeディレクティブ
  - Berkeley DBへの更新処理性能の向上
    - set_lg_bsizeディレクティブ
- サンプルファイルをコピーするだけ

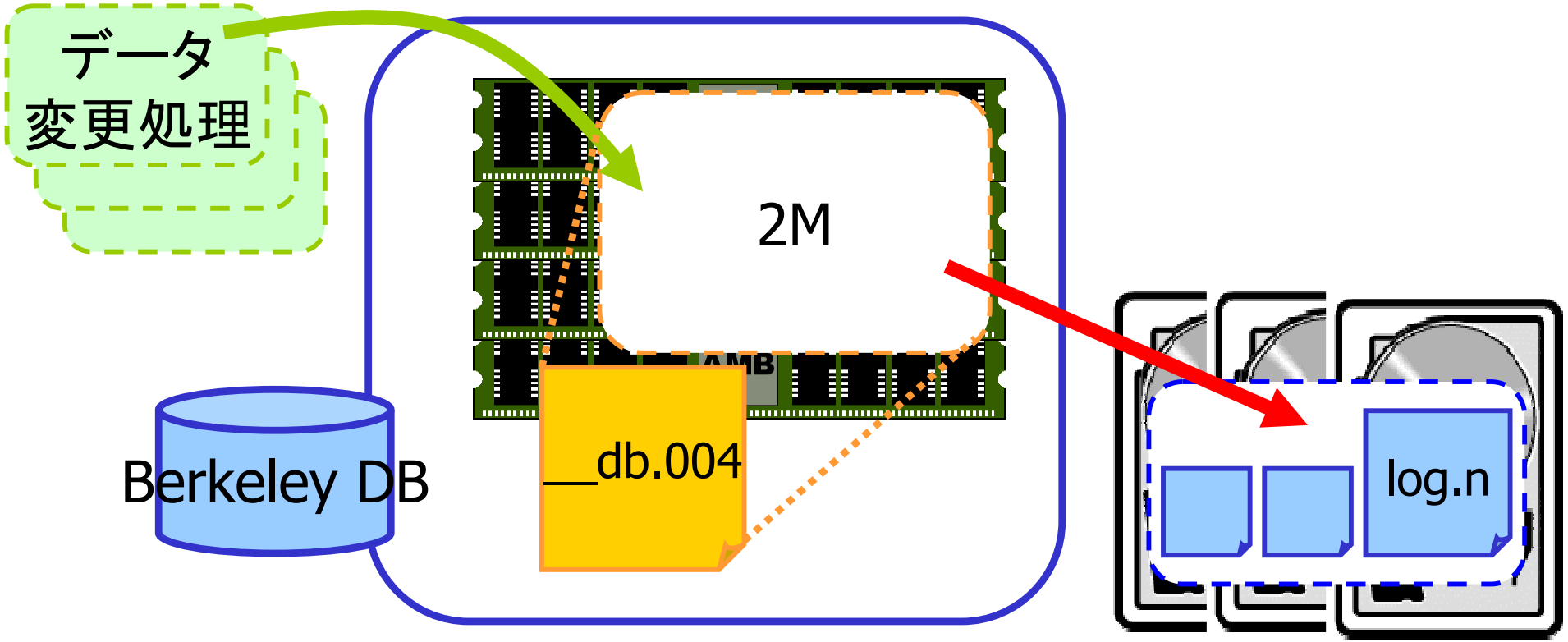
set_cachesizeディレクティブ

- Berkeley DBが共有メモリとして利用するバッファプールのサイズ
  - 目標は、*.bdbの合計サイズより大きく



set_lg_bsizeディレクティブ

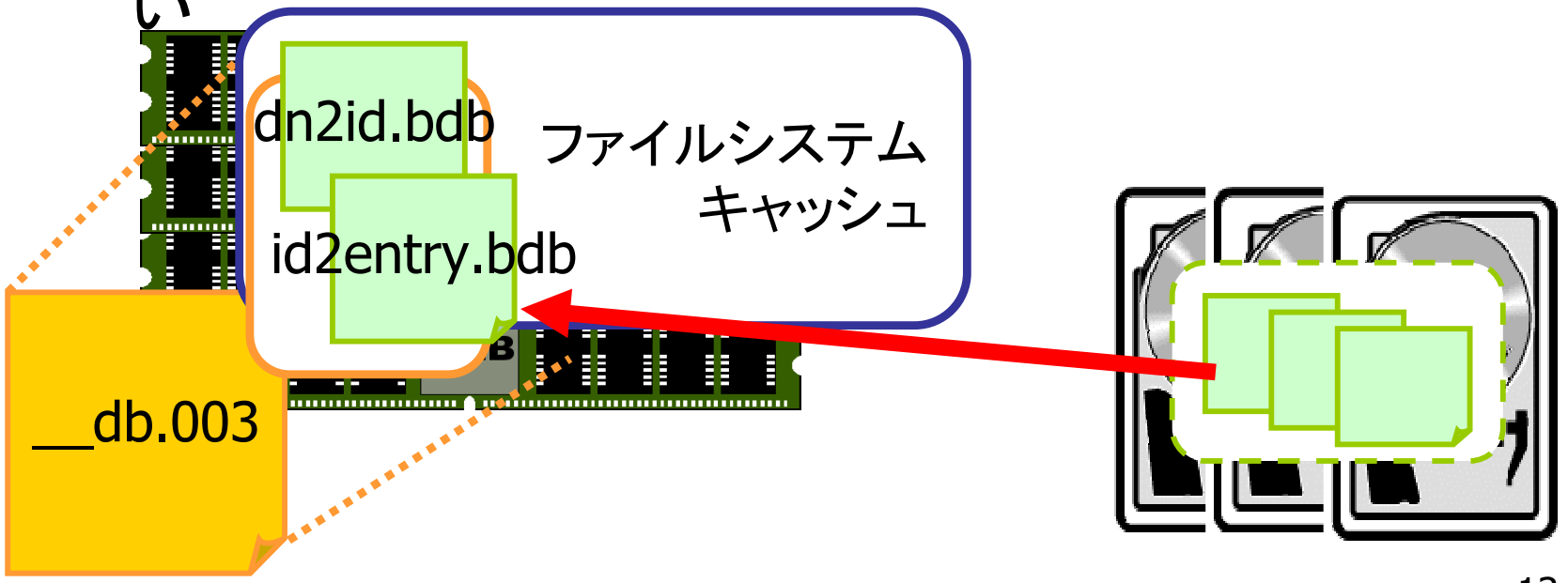
- Berkeley DBがログバッファとして利用するメモリ領域のサイズ
  - 大きな更新では、大きなサイズで性能が向上



Berkeley DB
バッファプール
で、その先へ！

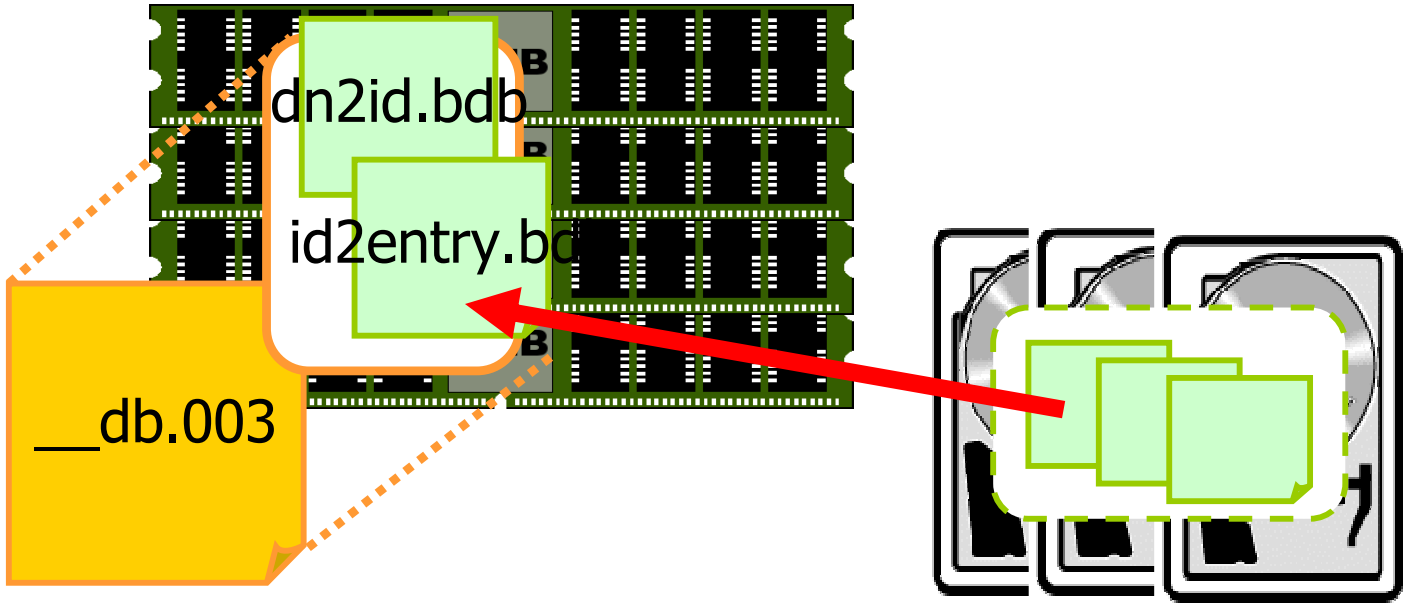
OS ファイルシステムキャッシュ

- Berkeley DB側で、OpenLDAPのデータをキャッシュしきれなくても、OSのファイルシステムキャッシュでキャッシュすることもできる
  - OSのキャッシュがあるため、Berkeley DB側のバッファプールのチューニング効果は見えにくいことが多い



Berkeley DB バッファプール

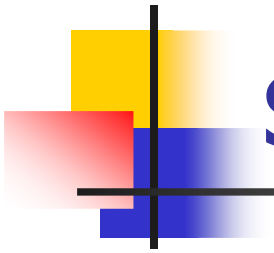
- OSのファイルシステムキャッシュを利用できない場合、Berkeley DBバッファプールが性能に影響
  - バッファプールへのページ読込み頻度が性能に影響
  - バッファプールのサイズはset_cachesizeで調整可能



バッファプールの利用価値

- Linuxでは、OS提供のファイルシステムキャッシュは、余剰メモリが割り当てられる
  - OSが、他のプロセスにも利用するキャッシュ
  - OpenLDAP側での制御ではない
    - OpenLDAPにとって、自分で管理できる Berkeley DBのバッファプールは、信頼しやすい
- 2重バッファを行わないために、わざわざO_DIRECTフラグを用いるという考え方も
  - しっかりとした、サイジングが必要

※ O_DIRECTフラグは、Linuxカーネル2.4.10以降で利用可能です。
OpenLDAPのバックエンドとなるBerkeley DBでの利用には、OpenLDAPコミュニティからのパッチと、コンパイルオプションの変更が必要です。

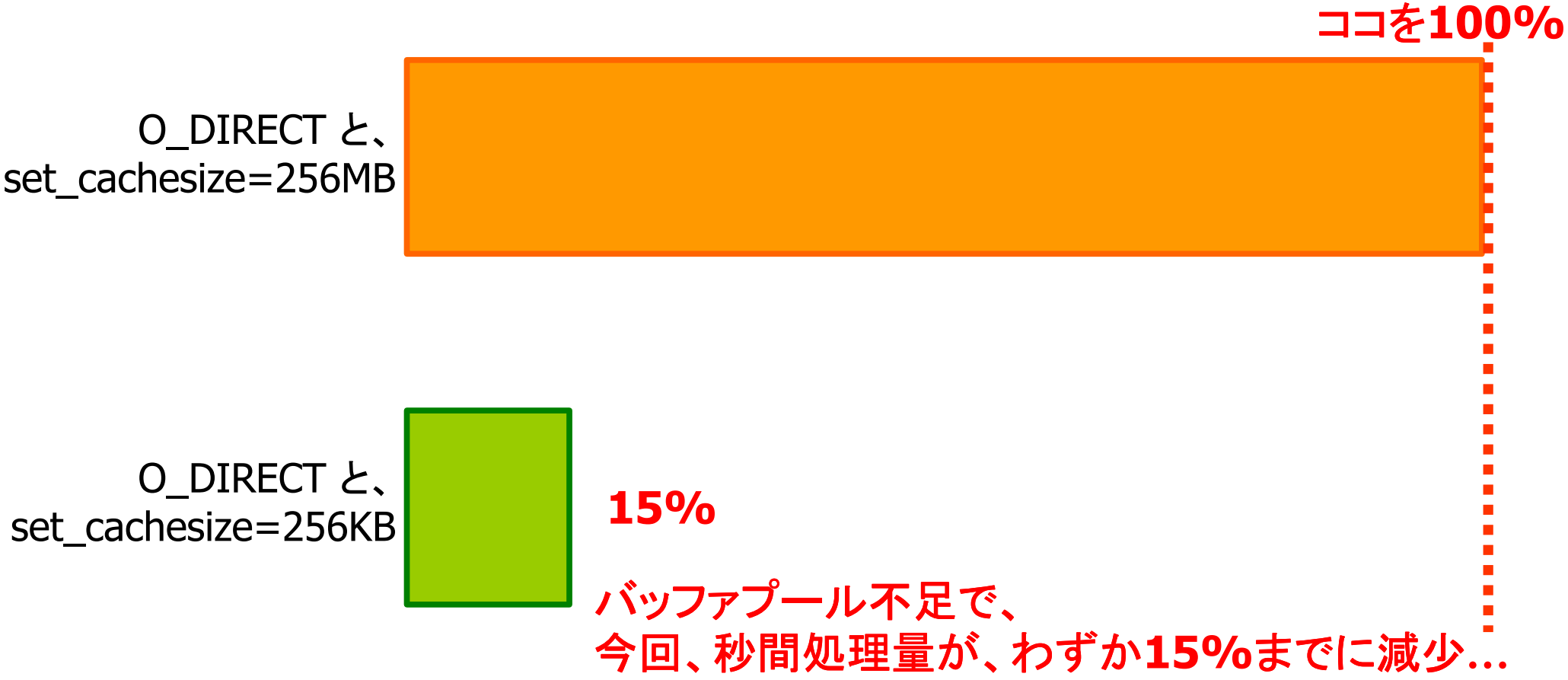


set_cachesizeディレクティブ

- デフォルトは、256KBytes
  - 最小は、20KBytes
- DB_CONFIGでは、256MBytesを設定
  - 500MBytes未満は、自動的に設定値+25%
  - __003.dbは、+25%で、320MBytesに
- ざっくり見積りは、OpenLDAPデータ領域で
  - # du -c -h *.bdb

Berkeley DBバッファプールの効果

- OSキャッシュが利用できない状況にする目的で、あえてO_DIRECTで.bdbファイルをオープン



※ 結果は、今回の検証環境での値です。性能劣化のインパクトは、Berkeley DB バッファプールへのページ読み込み処理の発生頻度にも関係します。

バッファプールで安心を

DB_CONFIG

set_cachesize

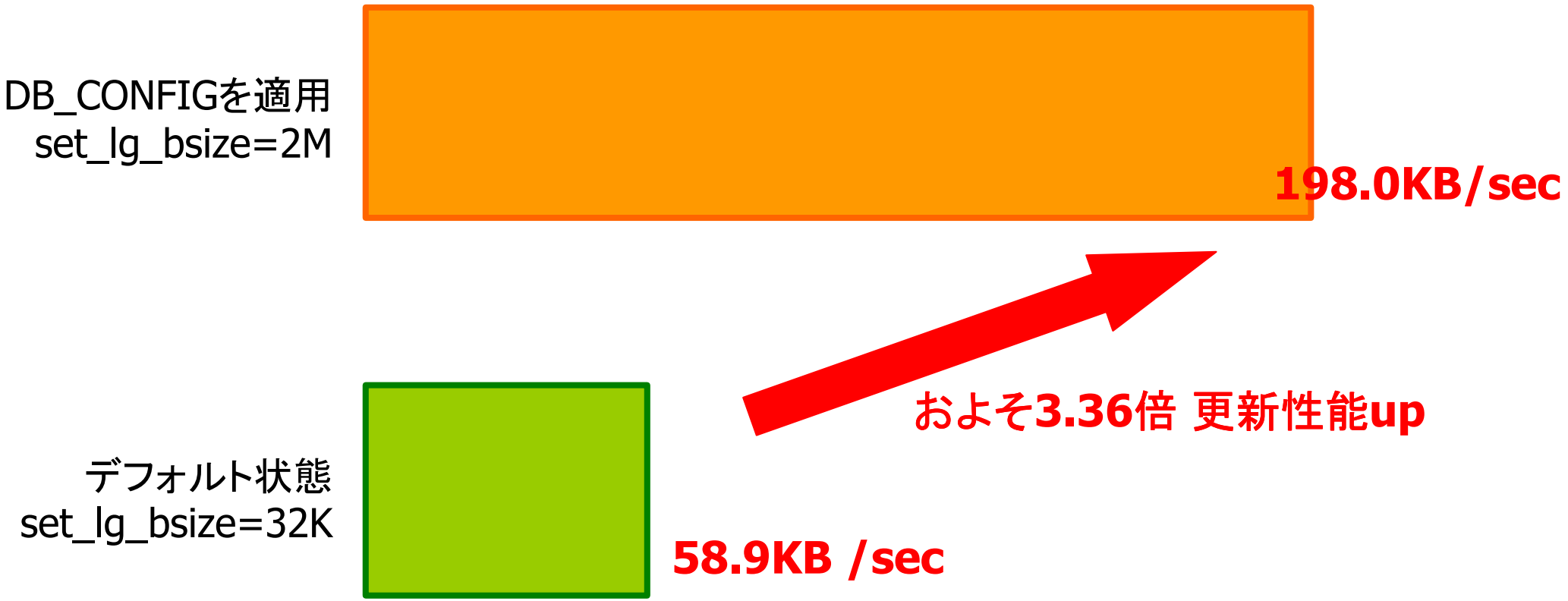
Berkeley DB の
バッファプールを活用しよう！

Berkeley DB
ログバッファ

で、その先へ！

slapadd でのデータロード速度

- DB_CONFIGで、ログバッファを調整



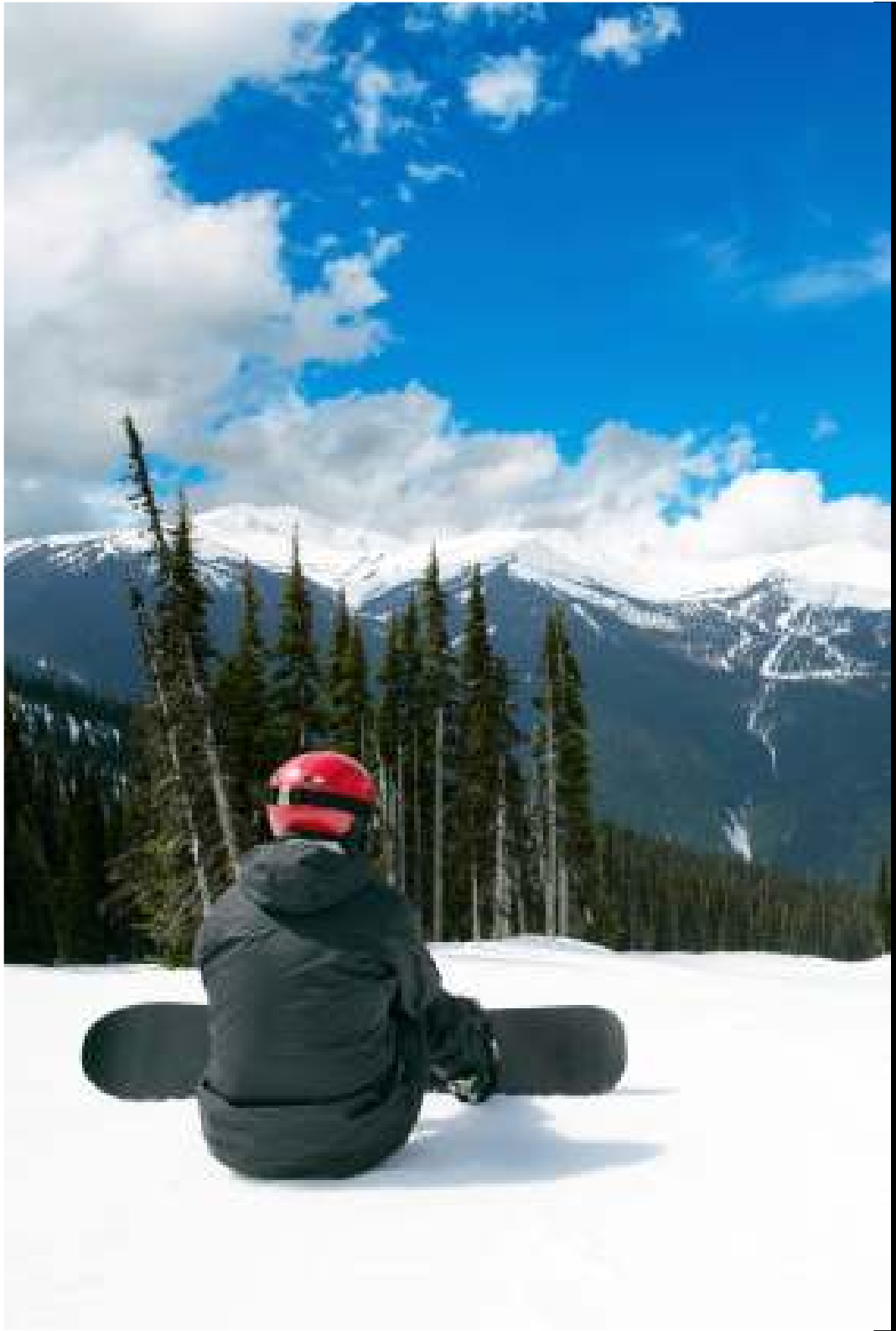
※ 結果は、今回の検証環境での値です。性能改善のインパクトは、ディスクIO性能にも依存します。

ログバッファで更新性能up！

DB_CONFIG

set_lg_bsize

Berkeley DB の
ログバッファを活用しよう！



検索の仕組み

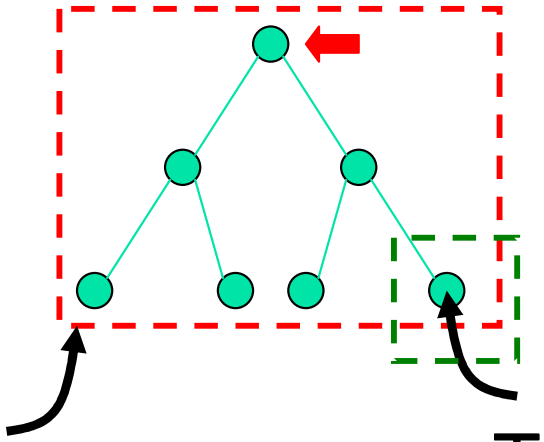
を知っておこう！

参考

ディレクトリ情報ツリーの検索

- ldapsearch -x -b dc=my-domain,dc=com -s sub

検索を開始するベース



検索スコープ

0：base (検索ベースのみ)

1：one (検索ベース以下1レベル)

2：sub (検索ベース以下全て)

3：children (検索ベースを除き以下全て)

のどれかを指定

ディレクトリ情報ツリー:

DIT (Directory Information tree)

エントリ

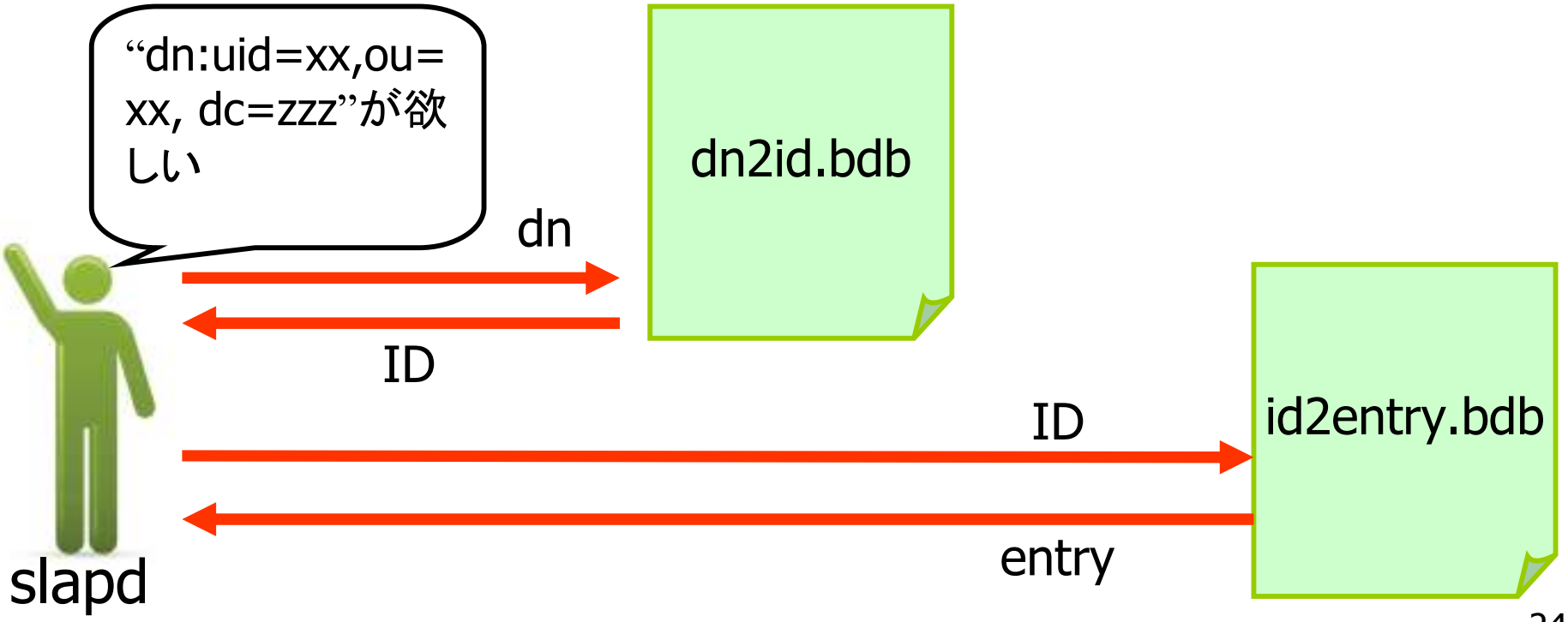
識別名: DN (Distinguished Name)

や、属性: (Attribute)を持つ

dn:uid=xx,ou=yy,dc=zz
cn:xx
uid:xx
sn:xx
ObjectClass:…
ObjectClass:…
…

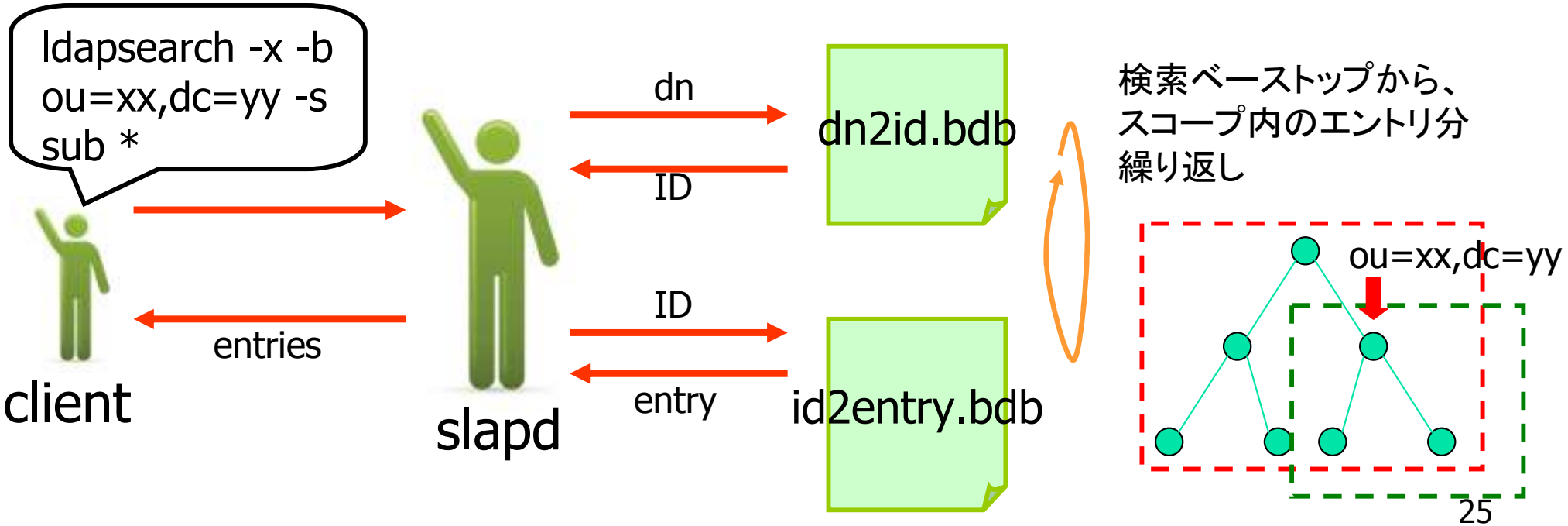
back-bdbでの、エントリの扱い

- Berkeley DBは、Key/Valueストレージ
  - リレーショナルDBではない
- OpenLDAPの各エントリに一意のIDを付与



back-bdbでの、エントリ検索

- 最初に、検索ベーストップのIDを取得
  - 続いて、検索ベーストップのエントリを取得
- 繰り返し、同じ要領で検索スコープ内のDNと、エントリを取得

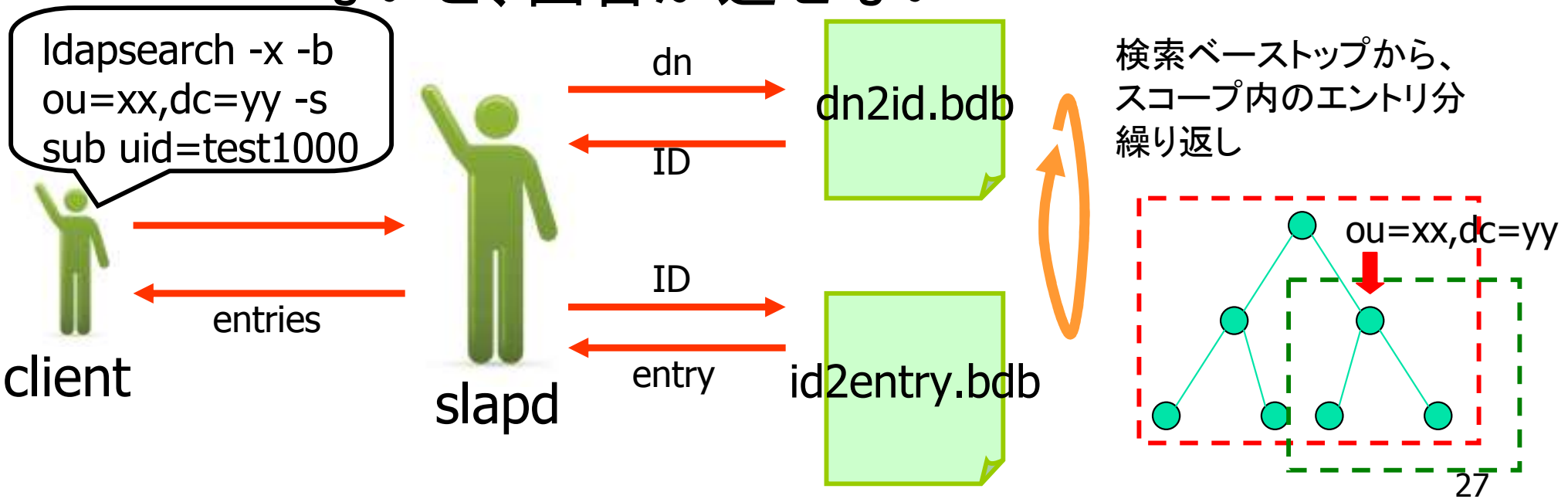


インデックス

で、その先へ！

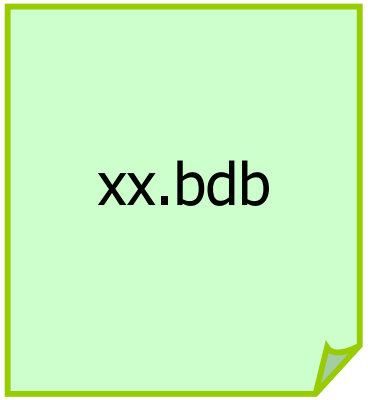
フィルタを付けてのエントリ検索

- 検索条件にフィルタを利用し、検索対象となる件数を減らしたいが… (ここでは、uid=test1000を指定)
  - フィルタと比較する情報は、id2entry.bdbに格納されている為、やっぱり何度もDBアクセスしないと、回答が返せない



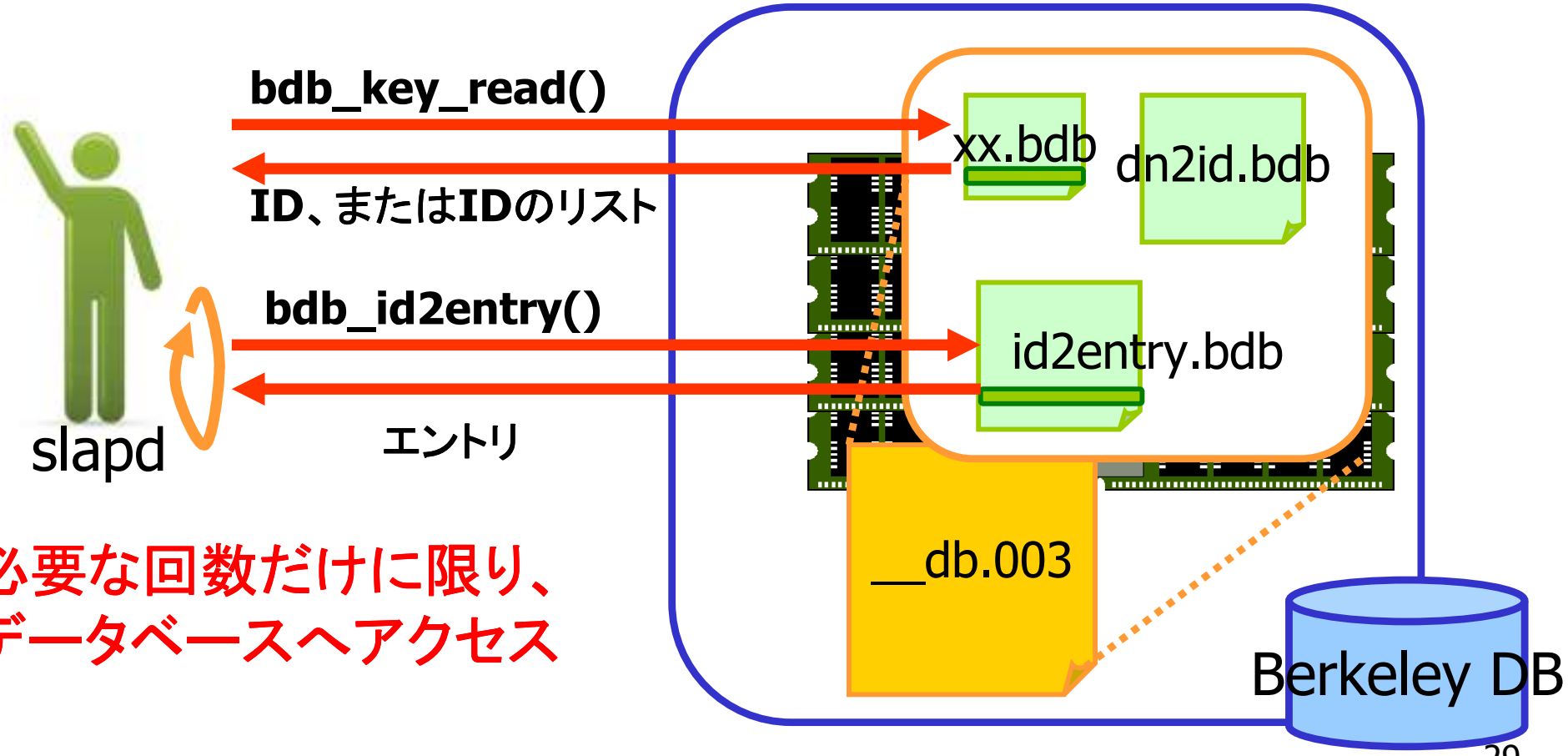
インデックスファイル

- 「インデックスキー - ID」をマップ
  - 与えられた検索フィルタから、必要エントリのみへのアクセスを可能にする情報
- dn2id.bdbは、「DN - ID」 をマップ
- id2entry.bdbは、「ID - エントリ」をマップ



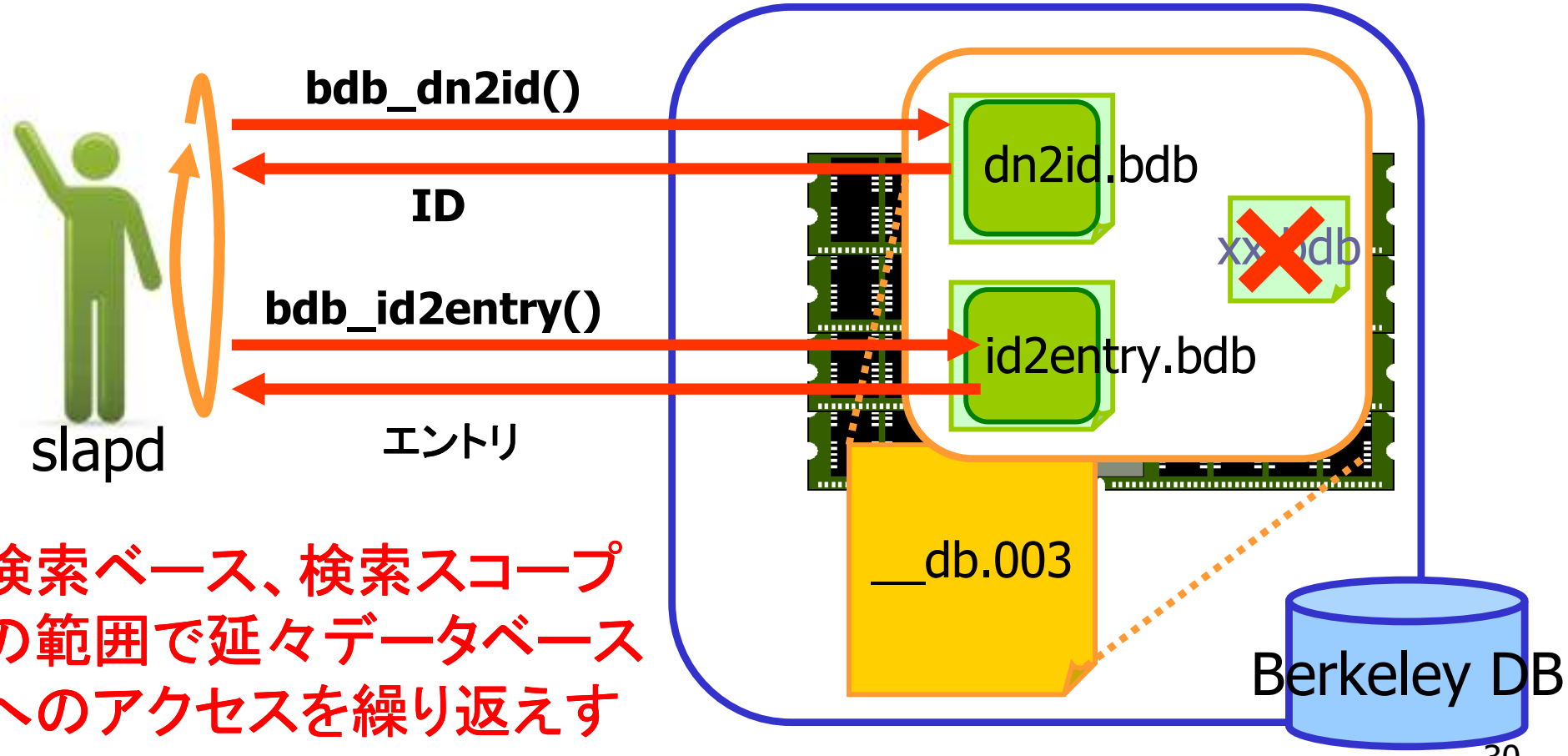
インデックスを利用した検索

- インデックスファイルを利用して、ピンポイントで必要なエントリを読む為のIDを取得



インデックスを利用しない検索

- ピンポイントのDBアクセスができず、延々とエントリを取得し続けなければならない

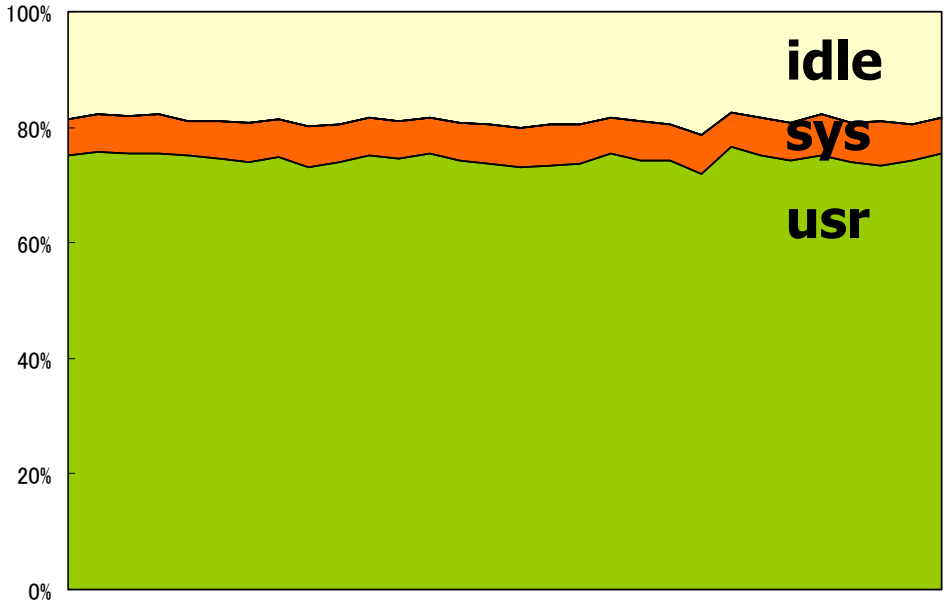


検索ベース、検索スコープの範囲で延々データベースへのアクセスを繰り返えす

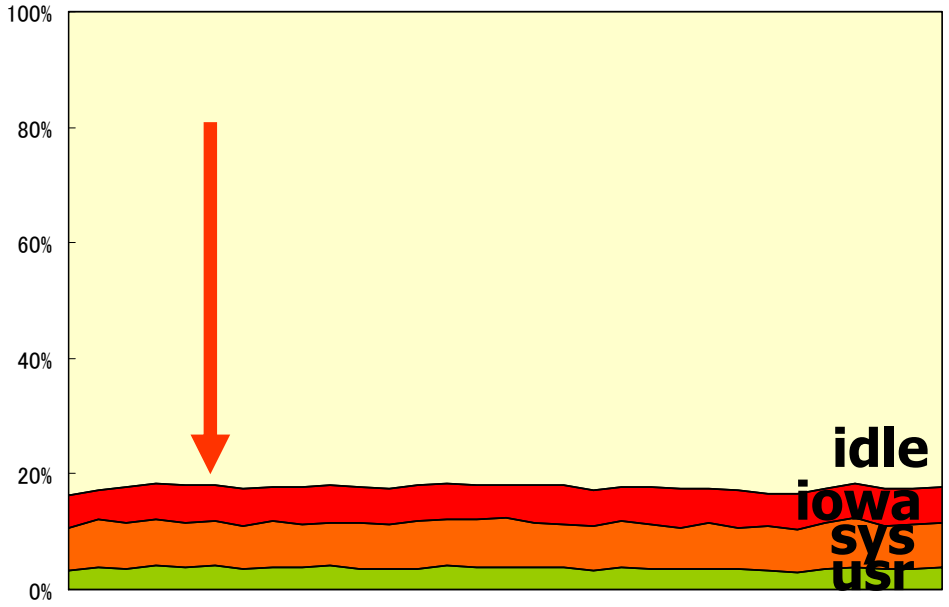
OSから見る、インデックス効果

- インデックス情報を利用できない場合は、slapdは、無駄に働きすぎてしまう
  - 頑張って働いても、結果は良くない

インデックスなしでの仕事量



インデックス利用時の仕事量



index ディレクティブの設定

- syslogに送られたログを参照することで、設定するべきインデックスを確認できる


```
# view servers/slapd/back-bdb/filterindex.c
…[略]…
if ( rc == LDAP_INAPPROPRIATE_MATCHING ) {
        Debug( LDAP_DEBUG_ANY,
            "<= bdb_equality_candidates: (%s) not indexed¥n",
            ava->aa_desc->ad_cname.bv_val, 0, 0 );
        return 0;
}
```

フィルタが指定されているのに、利用できなかったら、

slapd.confとかに、「loglevel 0」が指定されない限り、

この属性に、このタイプのインデックスを使いたかったとログに出すから…

※ loglevel 0 の場合は、出力されません。

index ディレクティブの設定

- syslogに送られたログを参照することで、設定するべきインデックスを確認できる


```
# tail -f /var/log/ldap.log
…[略]…
date time HOST slapd[PID]: <= bdb_equality_candidates: (uid) not indexed
…[略]…
date time HOST slapd[PID]: <= bdb_substring_candidates: (uid) not indexed
…[略]…
date time HOST slapd[PID]: <= bdb_approx_candidates: (uid) not indexed
…[略]…
```

uid属性に、eqインデックスが欲しい

uid属性に、subインデックスが欲しい

uid属性に、approxインデックスが欲しい

index ディレクティブの設定

- 同じ内容が頻繁にログ出力される場合は、特に効果が期待できる

slapd

slapd.conf

```
include …

database bdb
suffix dc=my-domain,dc=com
…[略]…

index uid eq

database bdb
suffix dc=example,dc=com
…[略]…
```

グローバルセクション

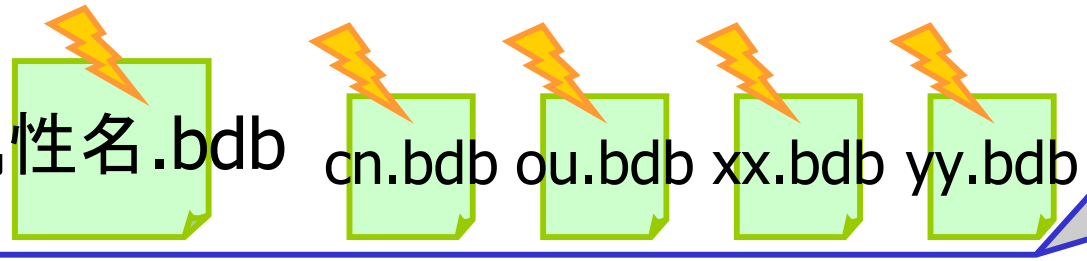
バックエンドDB #1

バックエンドDB #2

slapindexでのインデックス作成

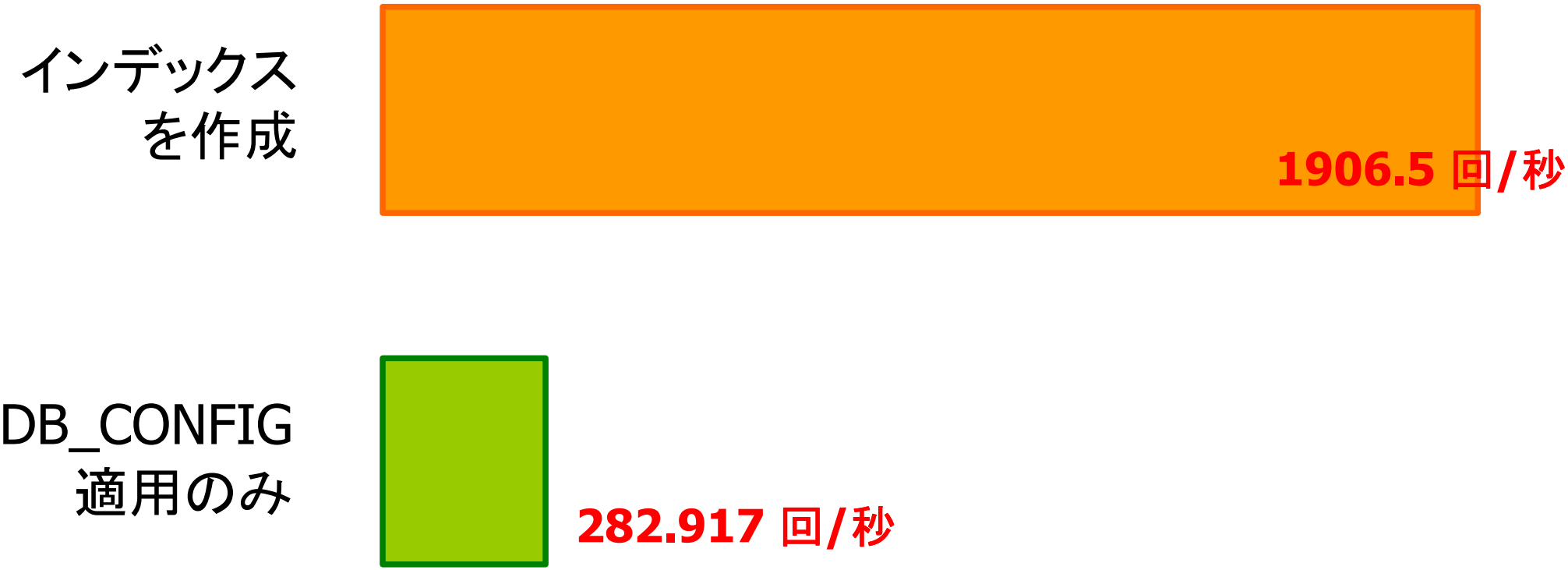
- slapindex でのインデックス作成手順
  1. OpenLDAPサーバの停止
  2. slapd.confの index を更新
  3. slapindexを実行

```
# slapindex -b dc=my-domain,dc=com  -v
indexing id=00000001
indexing id=00000002
indexing id=00000003
indexing id=00000004
…[略]…
```



インデックスの効果

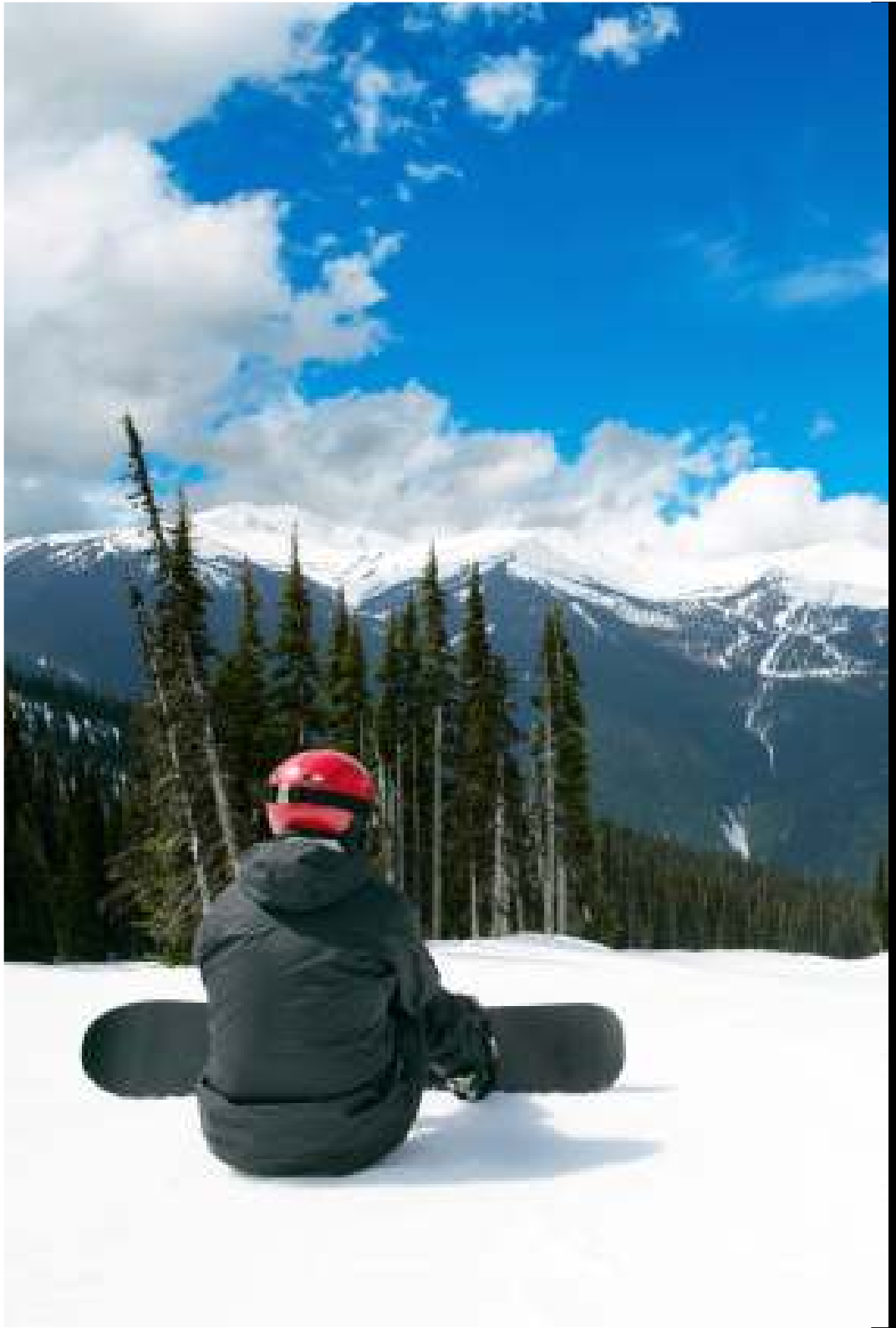
- インデックス検索で、6.7倍の性能改善



※ 結果は、今回の検証環境での値です。性能改善のインパクトは、検索対象となるエントリ数にも依存します。

インデックスで**6.7倍**

インデックスを活用しよう！



ログファイル

を知っておこう！

OpenLDAPのログファイル

今回の対象	ディレクティブ	役割
◎	loglevel	syslogに記録するログ。出力する内容(出力量)を調整できる
	logfile	loglevelで指定した内容を、syslogに加え、指定するファイルへも重複して出力させる
	overlay accesslog	OpenLDAPへの操作ログを、バックエンドデータベースへ蓄積する
	overlay auditlog	全ての変更情報を、指定するファイルに記録する
	DB_CONFIG ファイル set_lg_max	Berkeley DBのトランザクションログファイル

ログレベルの検討

- loglevelに指定する値

Level (10進数)	Level (16進数)	Level (文字列)	説明
-1		any	enable all debugging
0	0x0		no debugging
1	0x1	trace	trace function calls
2	0x2	packet	debug packet handling
4	0x4	args	heavy trace debugging (function args)
8	0x8	conns	connection management
16	0x10	BER	print out packets sent and received
32	0x20	filter	search filter processing

ログレベルの検討

- loglevelに指定する値

Level (10進数)	Level (16進数)	Level (文字列)	説明
64	0x40	config	configuration file processing
128	0x80	ACL	access control list processing
✓256	0x100	stats	stats log connections/operations/results
512	0x200	stats2	stats log entries sent
1024	0x400	shell	print communication with shell backends
2048	0x800	parse	entry parsing
16384	0x4000	sync	LDAPSync replication
32768	0x8000	none	only messages that get logged whatever log level is set

syslogd

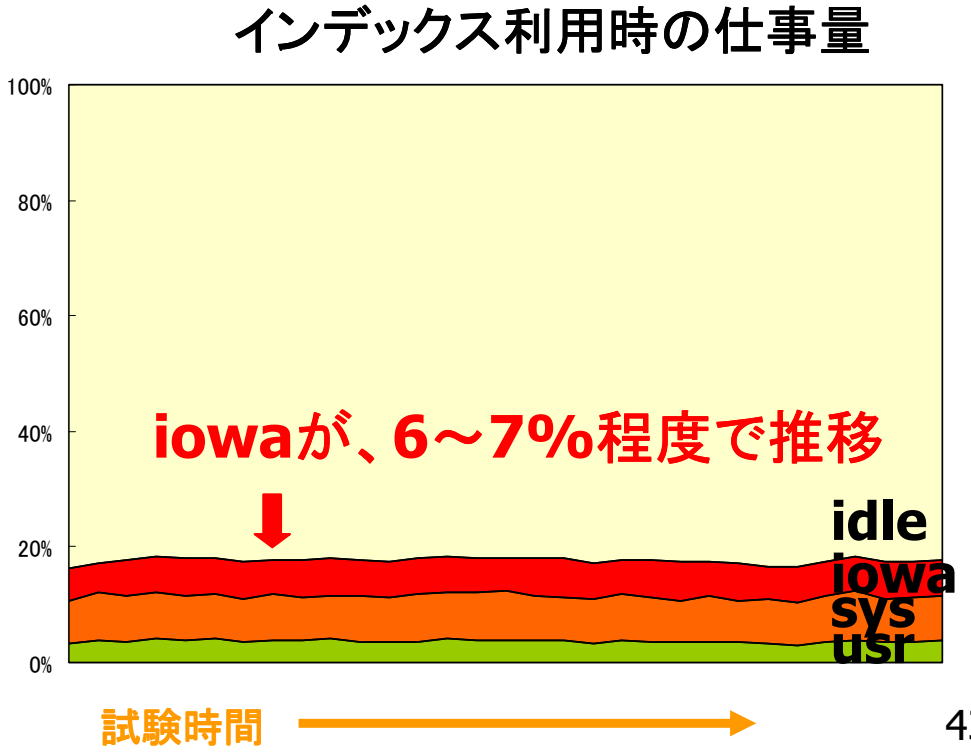
で、その先へ！

OSから見る、slapdの仕事ぶり

- インデックス作成後も、slapdは頑張ってる
  - しかし、CPU利用率のiowaitの割合からは、ディスクIOを待っている時間も長いことがわかる

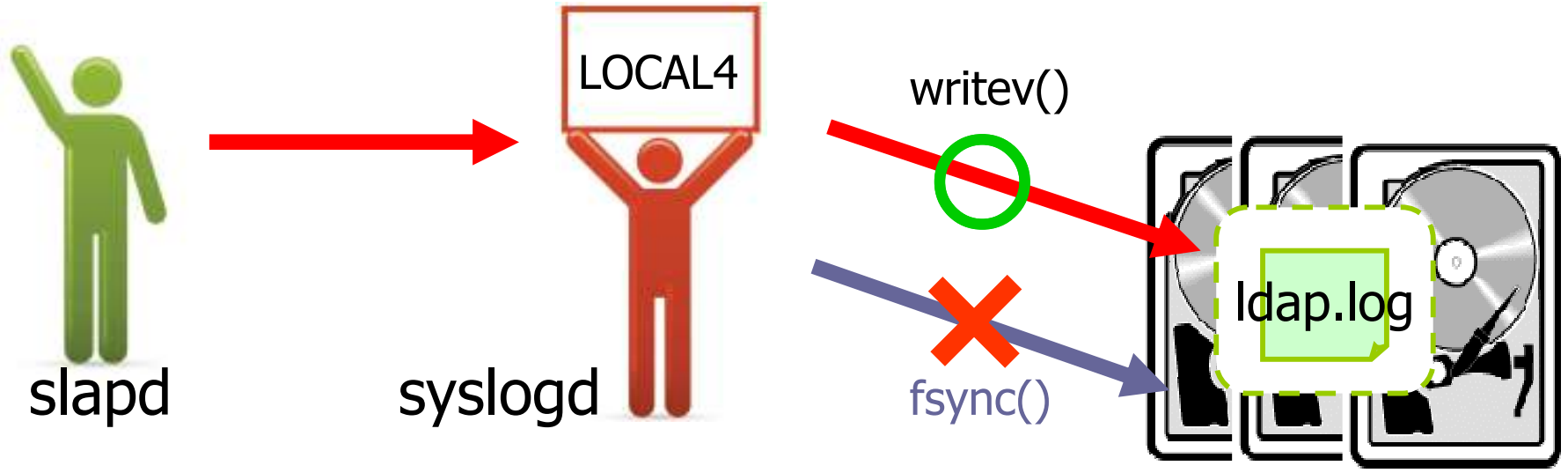
検索処理でも、管理者の為に、ログを出しています

ディスクIOは、コンピュータの処理の中で最も遅い処理



syslogの非同期書き込み

- syslogdがデータの書込み後に、ディスクへのフラッシュ(fsync)処理を行わない
  - 時間がかかるfsyncをスキップし性能を稼ぐ
  - 電源断時には、データを失う可能性がある



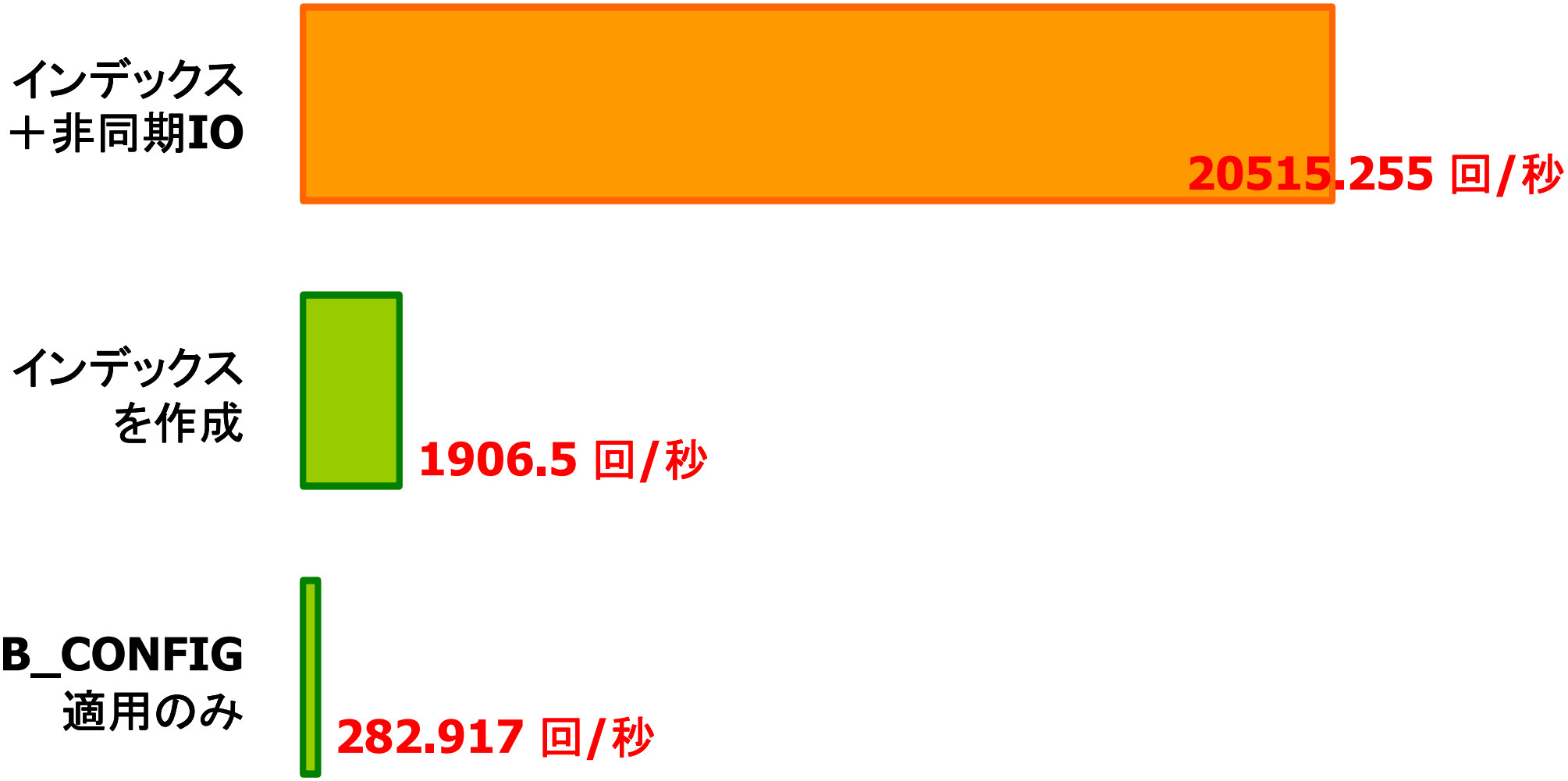
syslog非同期書き込みの設定

- OpenLDAPは、デフォルトでLOCAL4ファシリティを用いてログメッセージを送付
- syslog側で、LOCAL4ファシリティログを受け取り非同期で書込む設定、「-」が必要
  - # vi /etc/syslog.conf

```
…[略]…
local4.*                        -/var/log/ldap.log
…[略]…
```

  - syslog 再起動

syslog非同期書き込みの効果

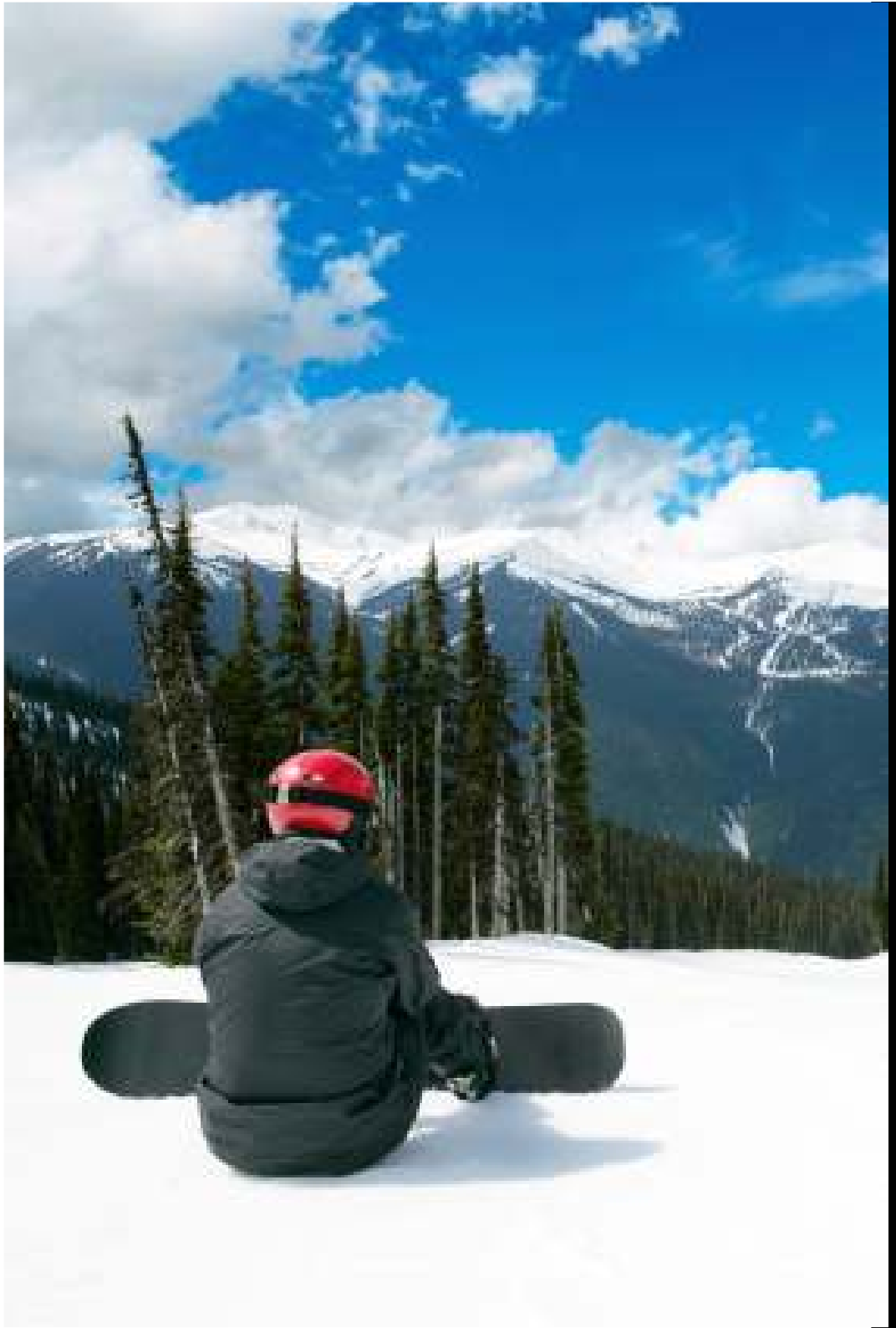


※ 結果は、今回の検証環境での値です。性能改善のインパクトは、ディスクIO性能、ログ出力量にも依存します。

10倍 UP！

非同期IOで、**10倍**

syslogdの
非同期書き込みを検討しよう！



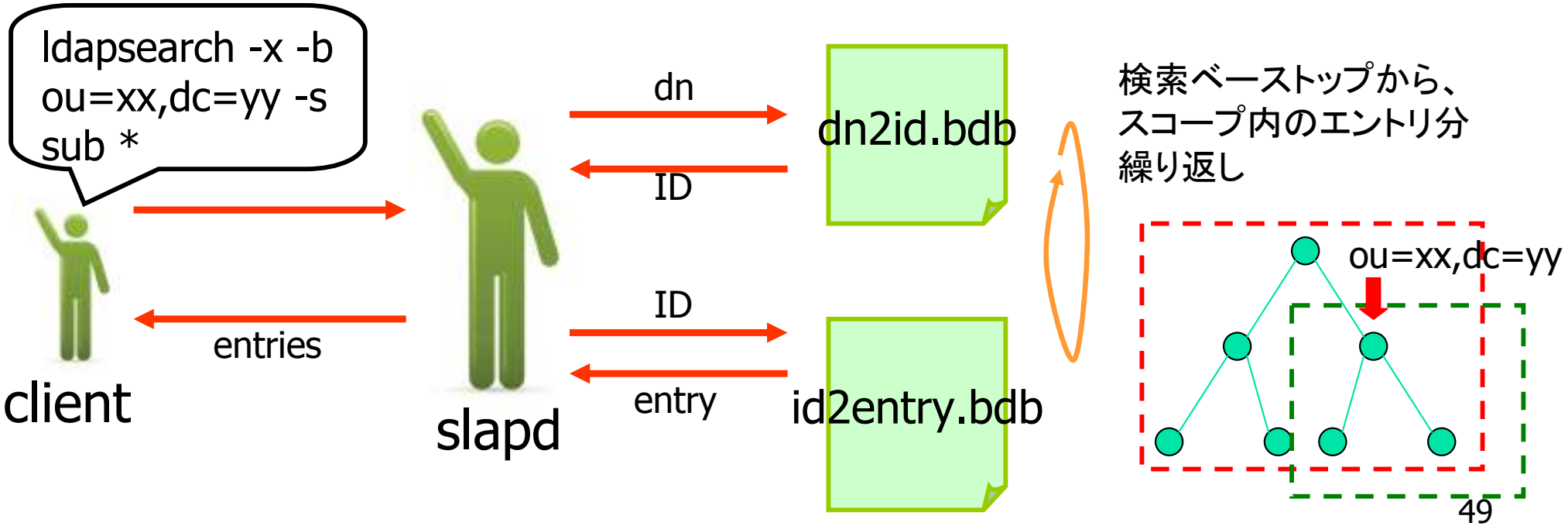
エントリキャッシュ

を知っておこう！

参考

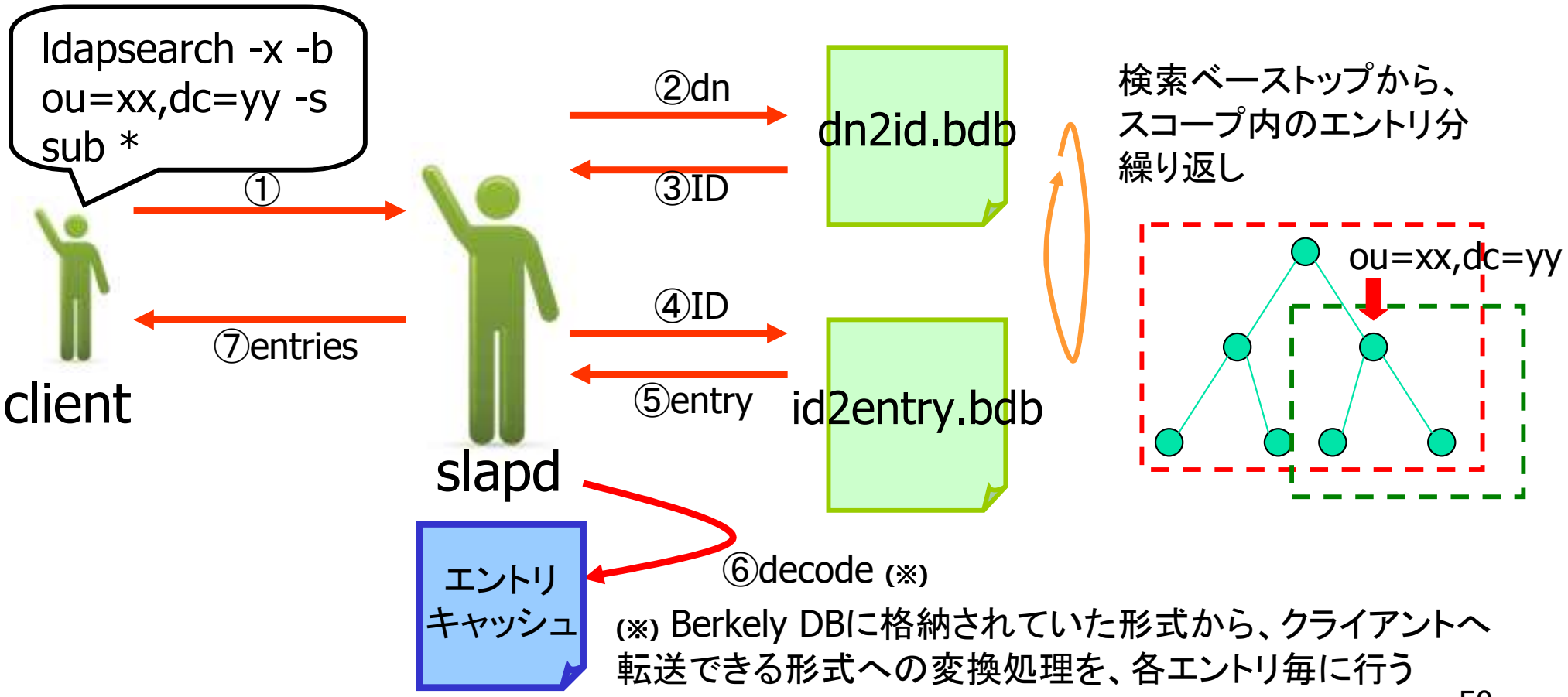
back-bdbでの、エントリ検索

- 最初に、検索ベーストップのIDを取得
  - 続いて、検索ベーストップのエントリを取得
- 繰り返し、同じ要領で検索スコープ内のDNと、エントリを取得



エントリデータのキャッシュ

- クライアントに回答する前にdecodeが必要
  - ②～⑥までの処理の繰り返し

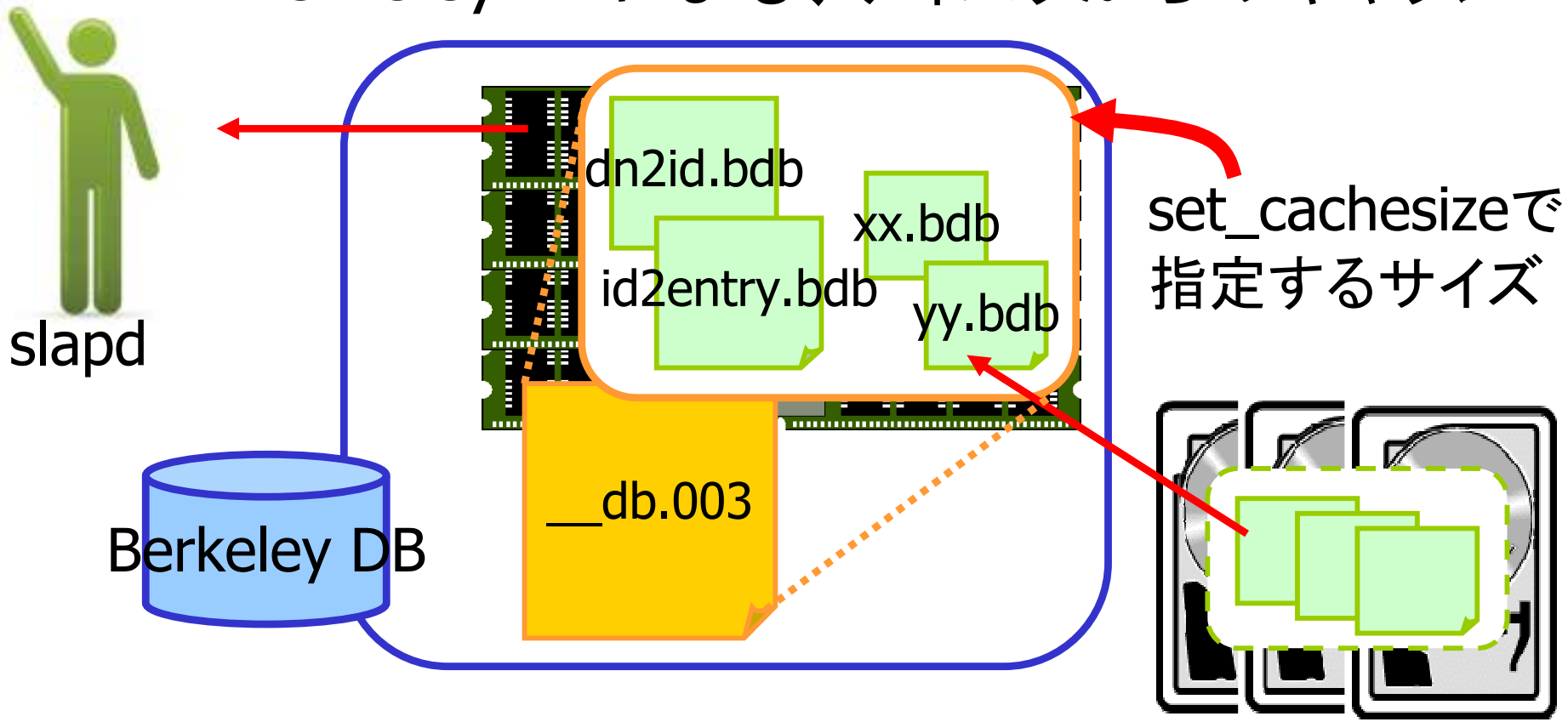


(※) Berkely DBに格納されていた形式から、クライアントへ転送できる形式への変換処理を、各エントリ毎に行う

参考

DB_CONFIG、set_cachesize

- DB_CONFIGに設定するキャッシュは、Berkeley DBが利用するバッファサイズ
  - Berkeley DBによる、ディスクからのキャッシュ

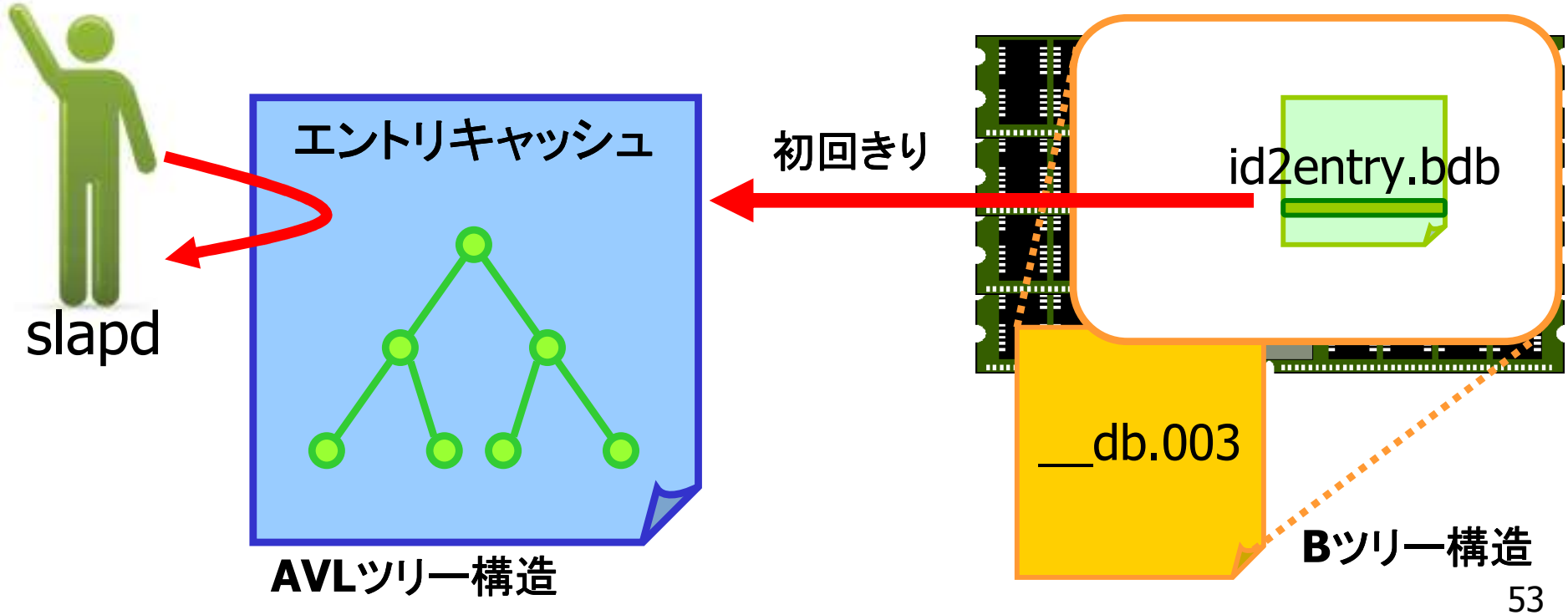


エントリキャッシュ

で、その先へ！

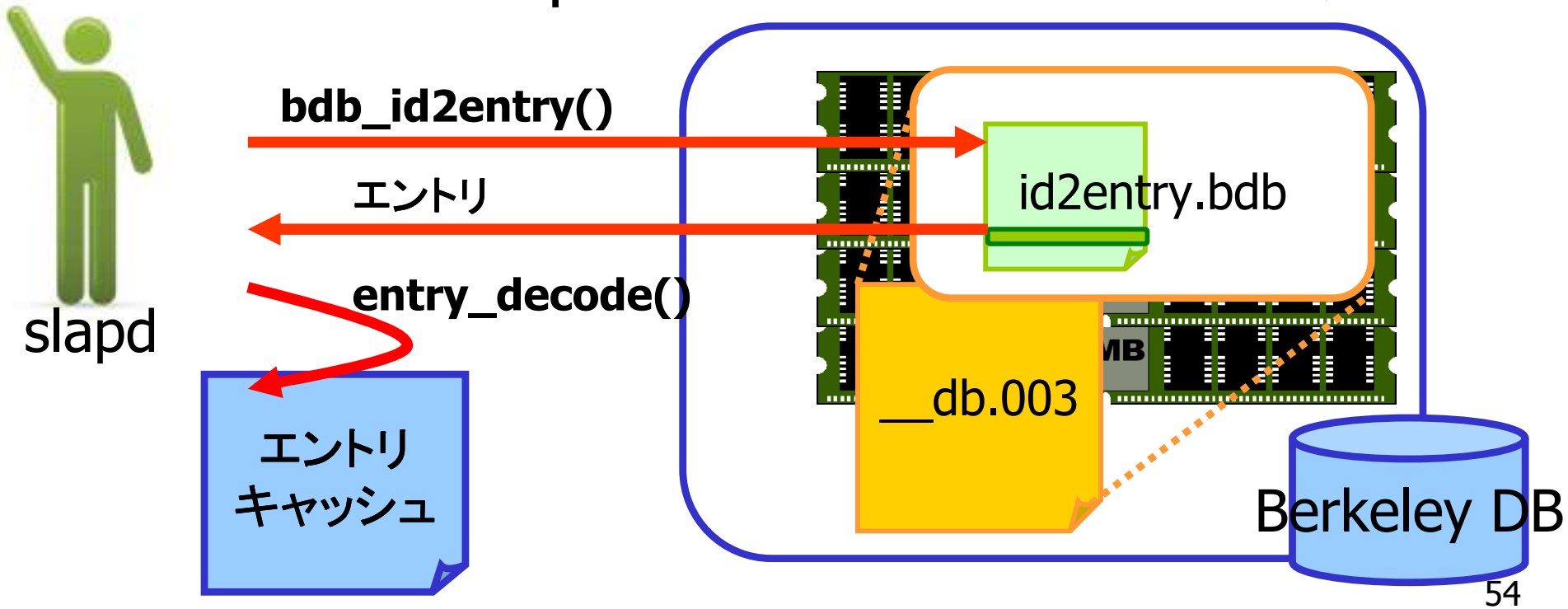
エントリキャッシュ

- 一度、Berkeley DBから取得、デコードしたエントリをOpenLDAP側でキャッシュ
  - 2回目以降、slapdはエントリキャッシュを利用して、DBアクセス、デコード処理を省略



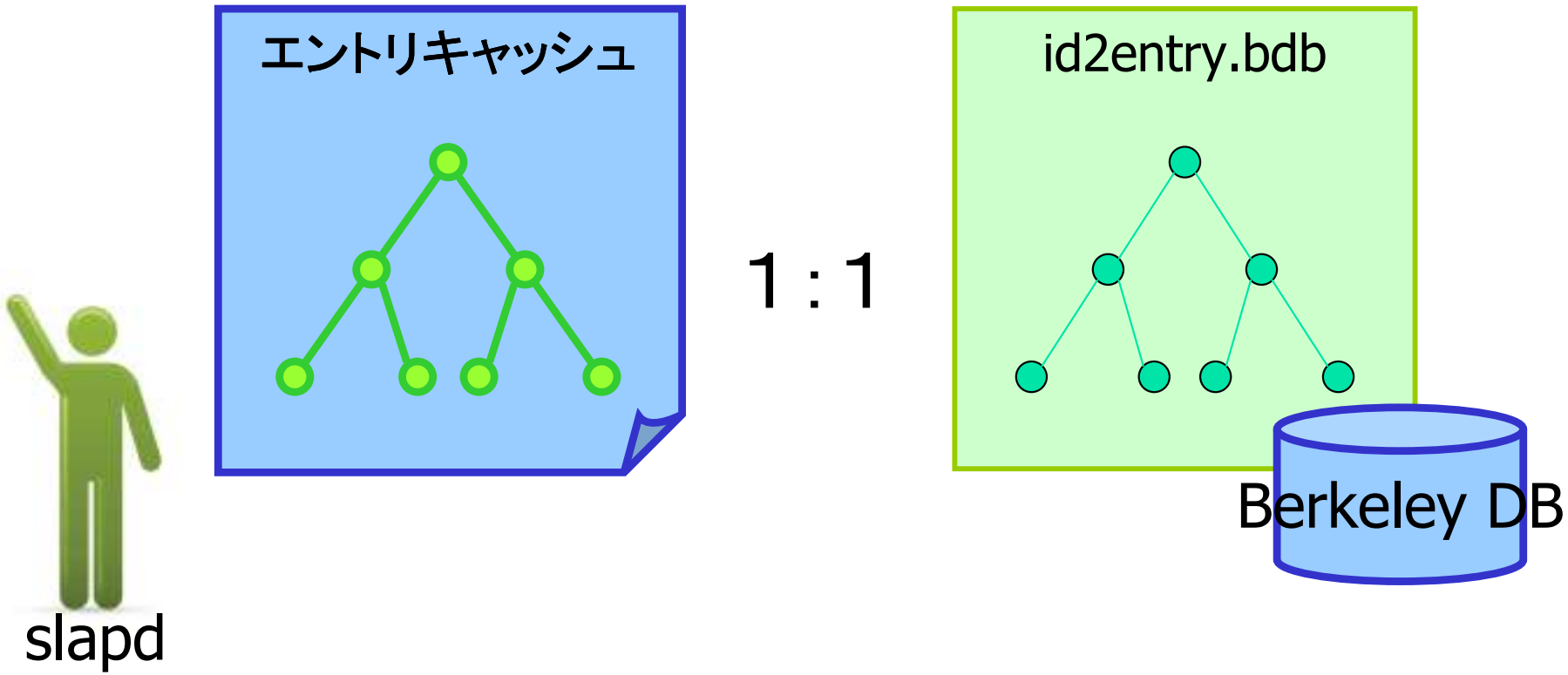
cachesizeディレクティブ

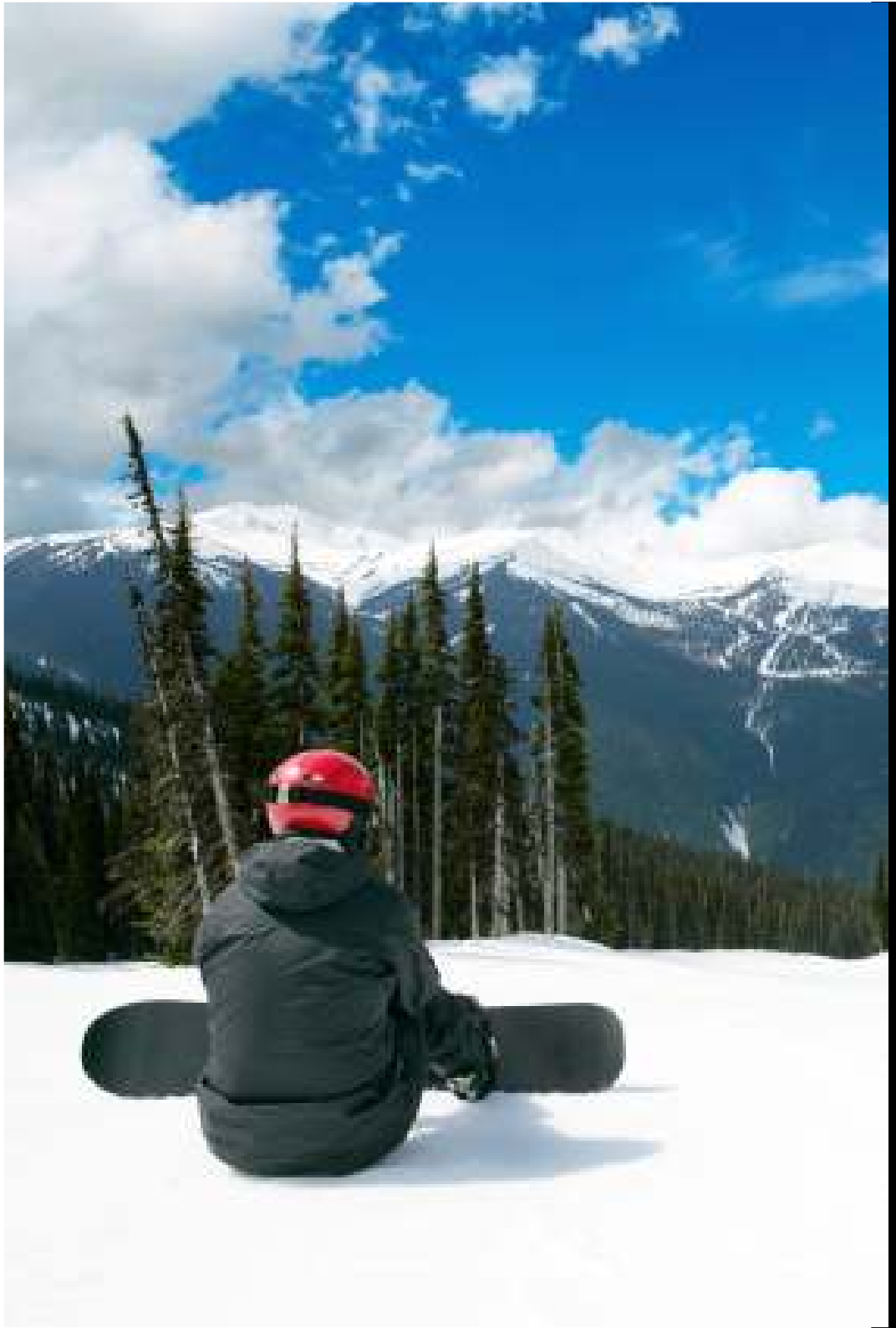
- バックエンドにBerkerey DBを利用している場合に、利用可能なディレクティブ
  - Berkeley DBから取得済み、デコード済みエントリを、slapd側でキャッシュする最大数を指定



cachesizeディレクティブ

- デフォルトは、1000
  - ベストは、エントリ数と同じ数だけ設定





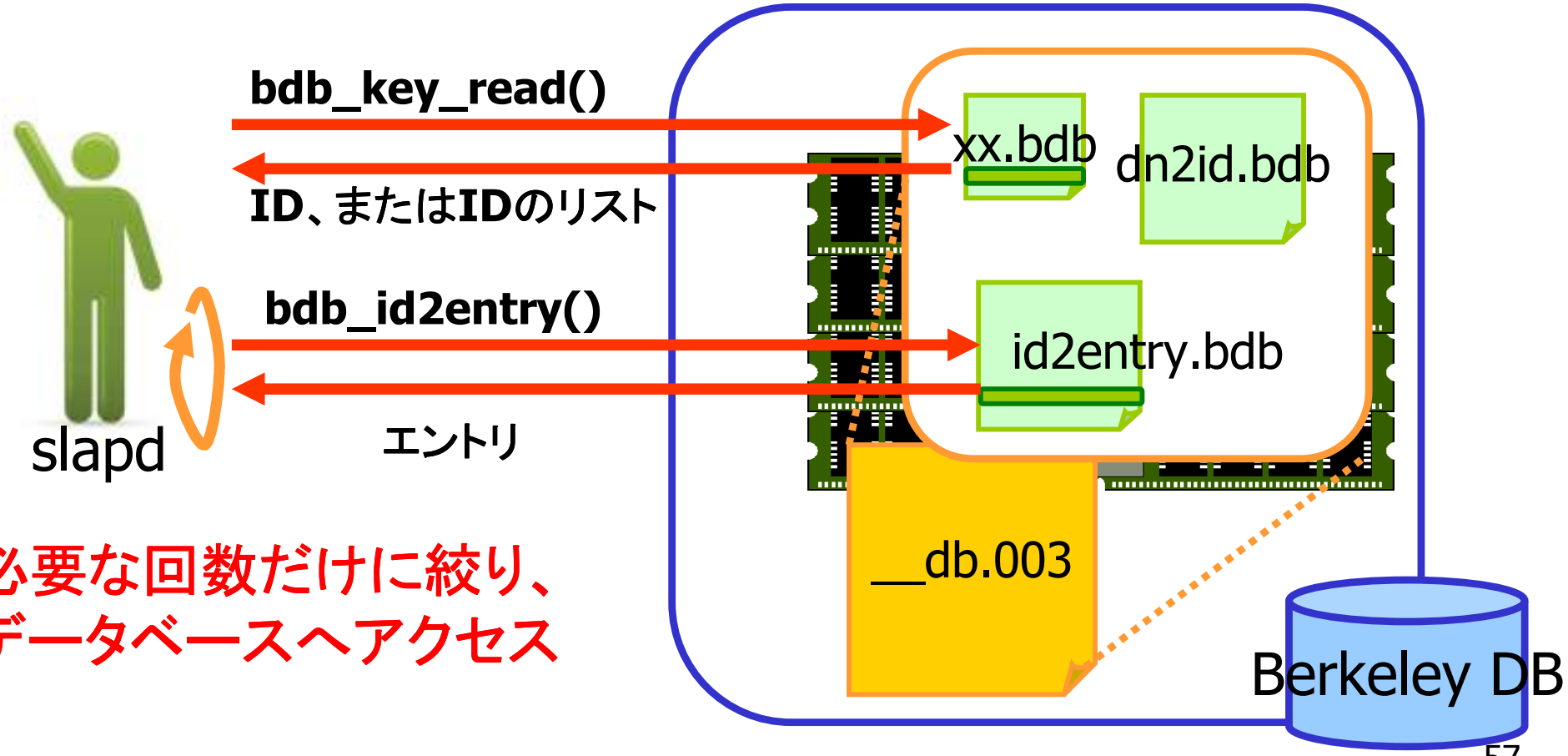
ID リストキャッシュ

を知っておこう！

参考

インデックスを利用した検索

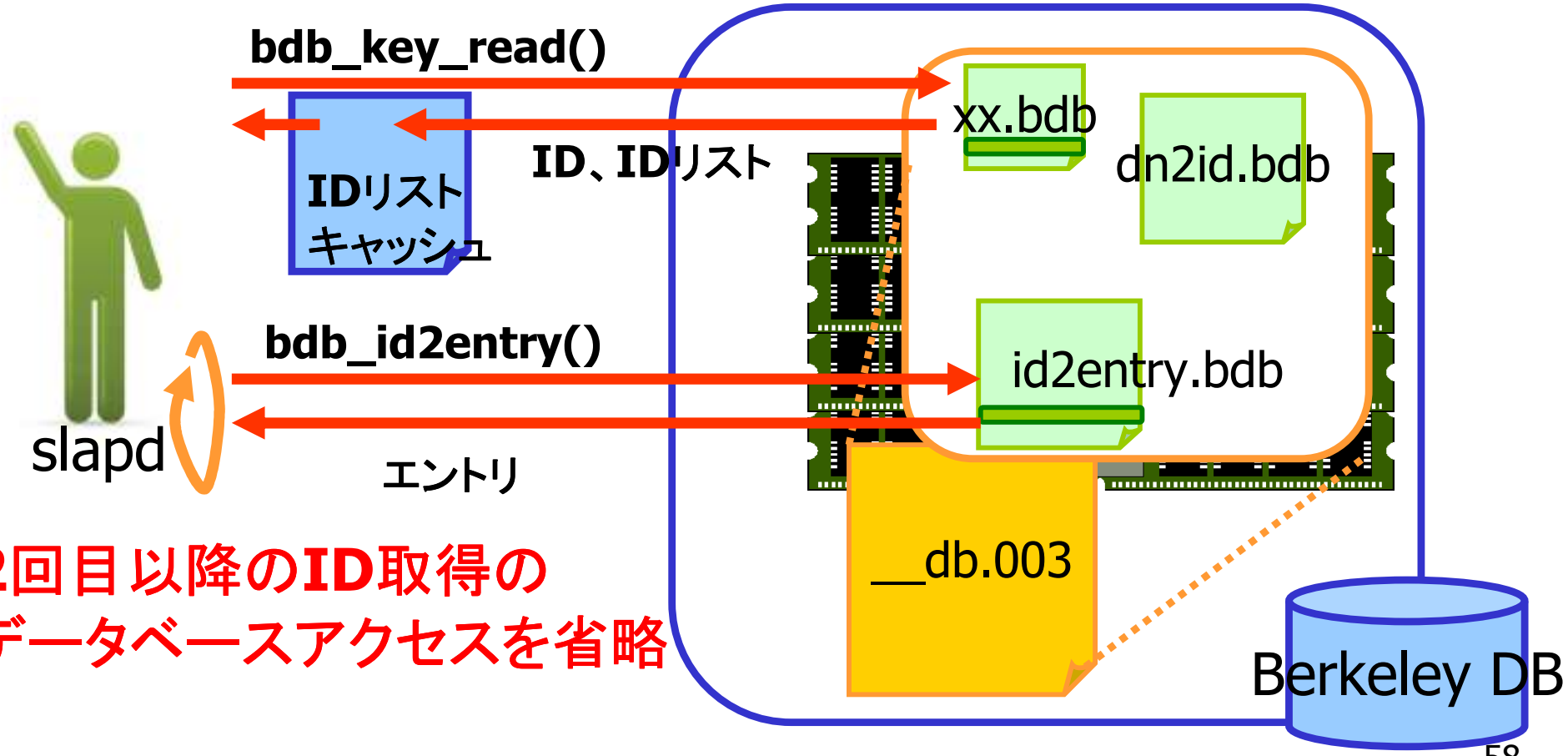
- インデックスファイルを利用して、ピンポイントで必要なエントリを読む為のIDを取得



必要な回数だけに絞り、
データベースへアクセス

キーでヒットしたIDのキャッシュ

- インデックスファイルから取得したIDを、OpenLDAP側でキャッシュすることが可能



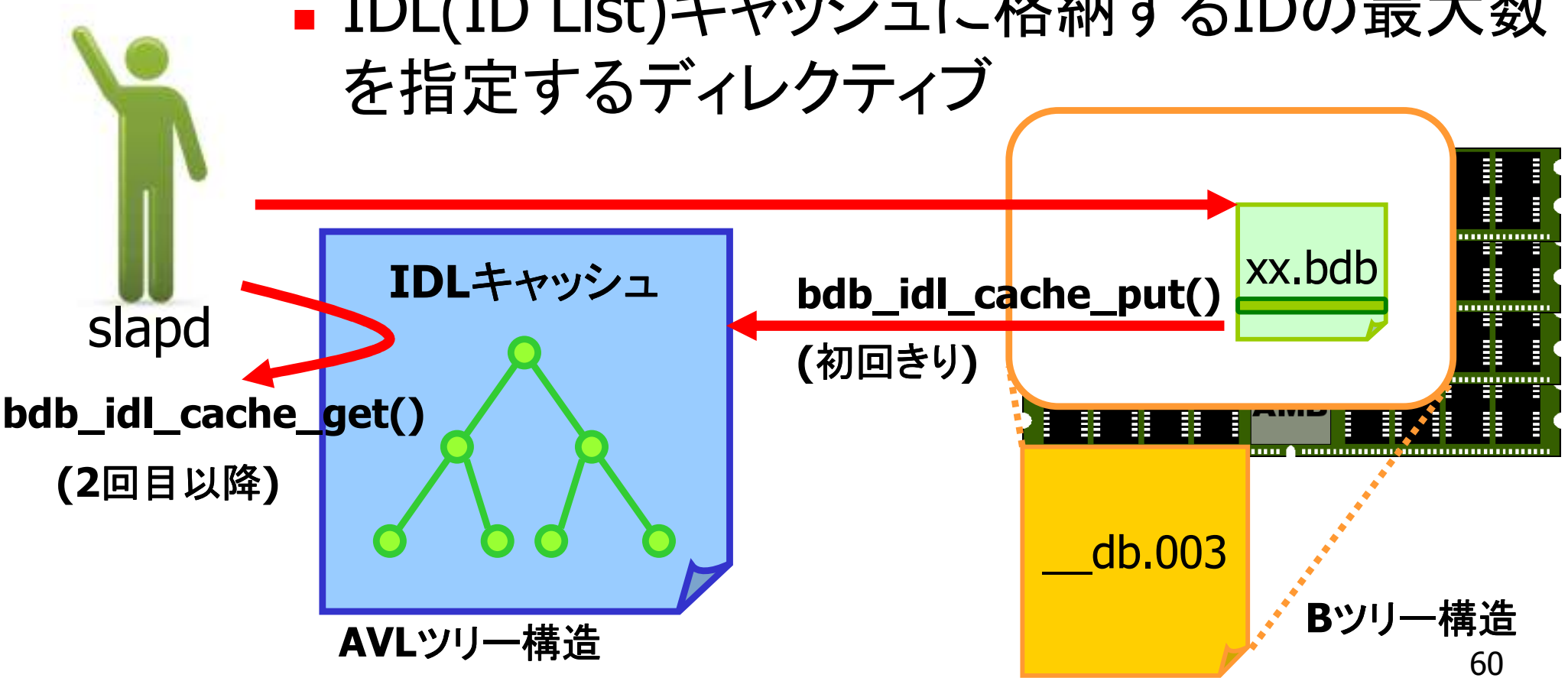
2回目以降のID取得の
データベースアクセスを省略

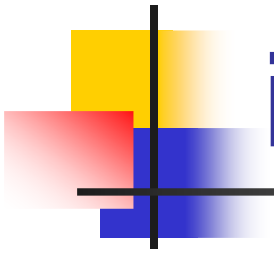
IDリストキャッシュ

で、その先へ！

IDリストキャッシュ

- バックエンドにBerkerey DBを利用している場合に設定可能
  - IDL(ID List)キャッシュに格納するIDの最大数を指定するディレクティブ

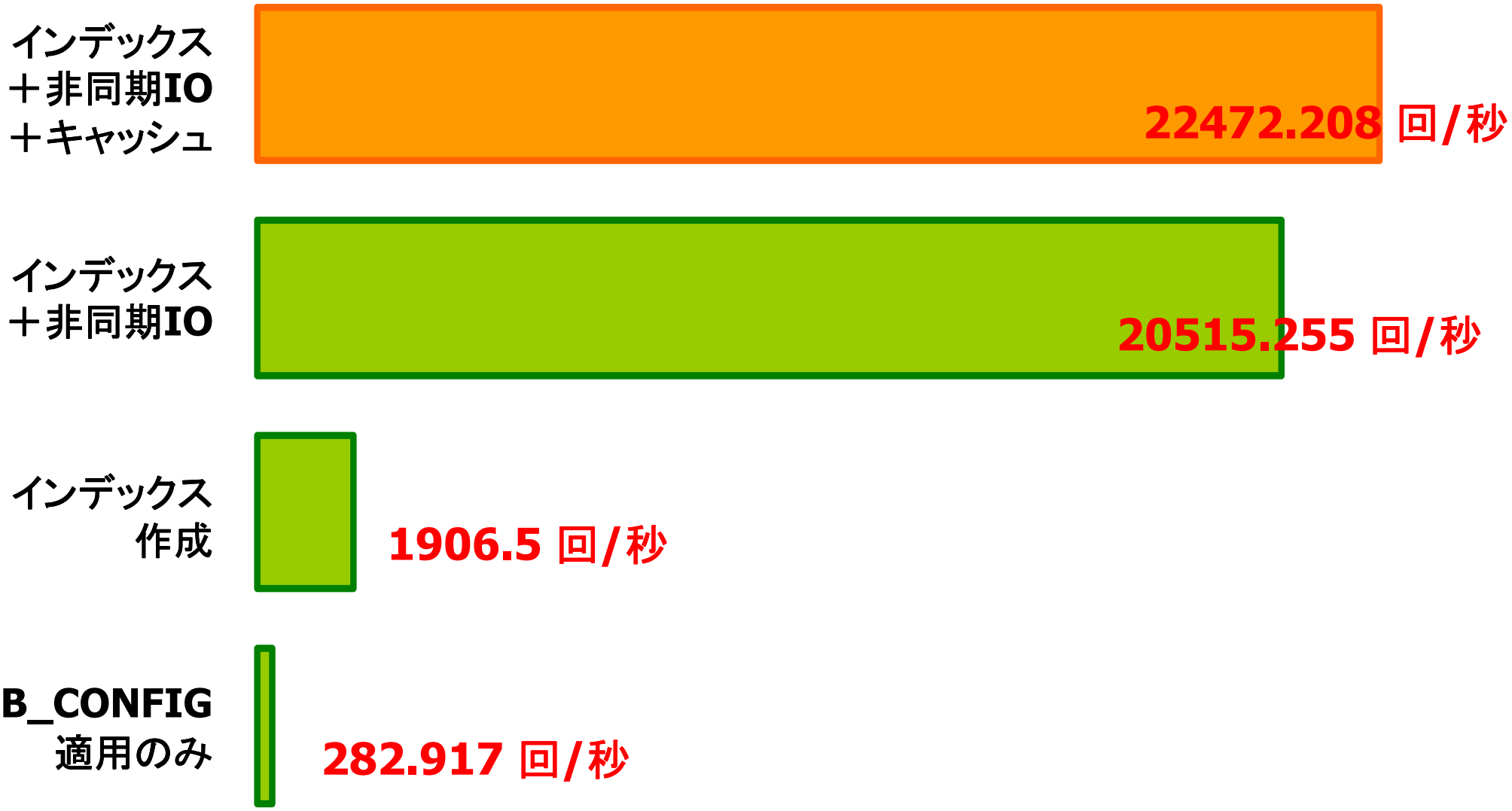




idlcachesizeディレクティブ

- デフォルトは、0。
  - IDLキャッシュは使わない
- BDBを利用している場合の設定目安
  - Back-bdbは、1エントリと同数
  - Back-bdbは、「1エントリにつき、1 ID」の関係
- HDBを利用している場合の設定目安
  - Back-hdbはIDを多く使う為、エントリ数の3倍

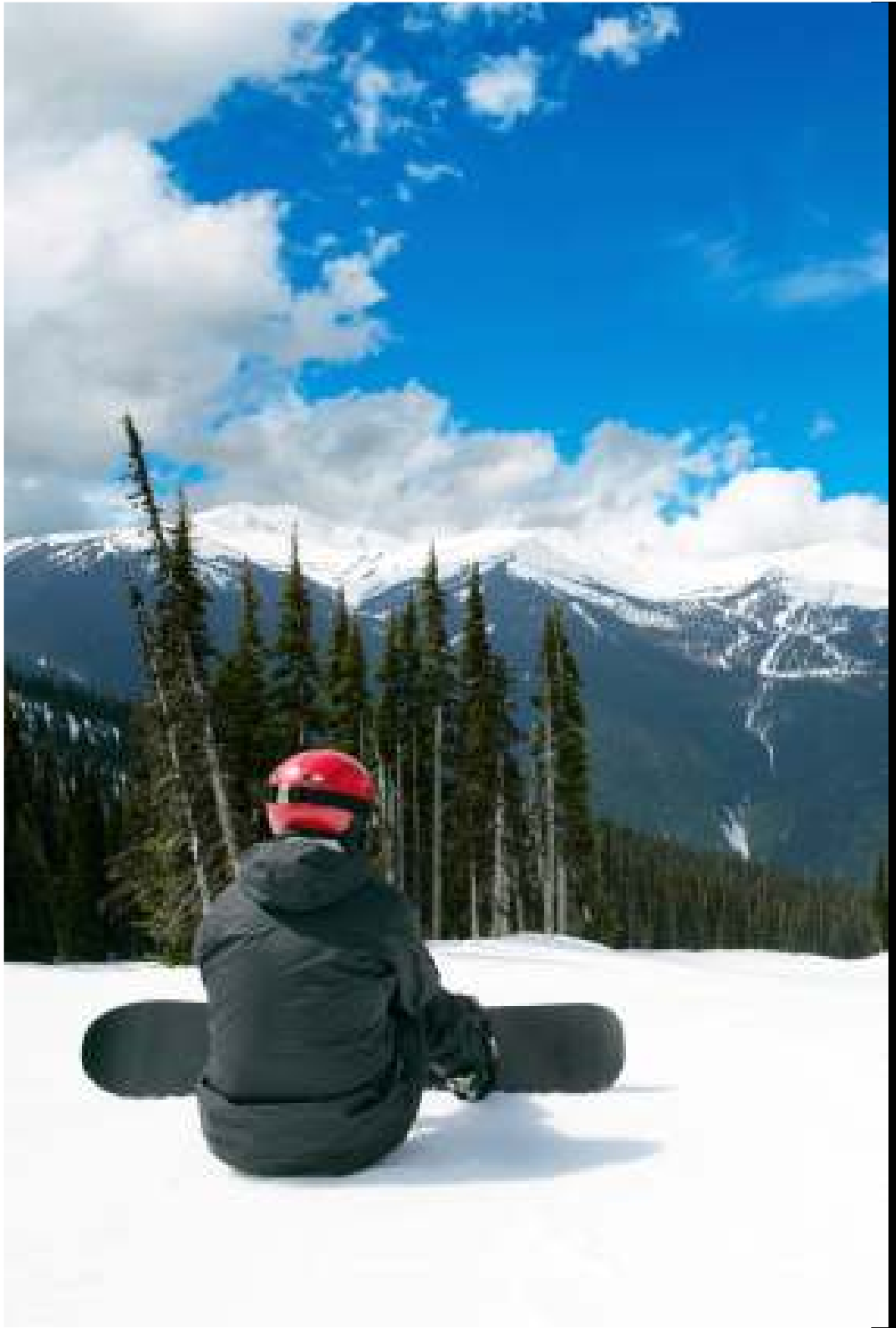
OpenLDAP側でのキャッシュ効果



※ 結果は、今回の検証環境での値です。性能改善のインパクトは、参照系処理、更新系処理の割合にも依存します。

キャッシュで、**10%**

OpenLDAP側での
キャッシュ技術を活用しよう！



スレッド処理

を知っておこう！

OpenLDAPが生成するスレッド

- OpenLDAPは、マルチスレッドで動作
- slapdは、リスナーと、ワーカーの2タイプのスレッドを生成する

スレッドが生成されるタイミング

- slapd起動時
  - slapdプロセスは、リスナースレッドを生成

- LDAPクライアントからのアクセス時
  - リスナーに加え、必要なワーカーをプール開始

参考

concurrencyディレクティブ

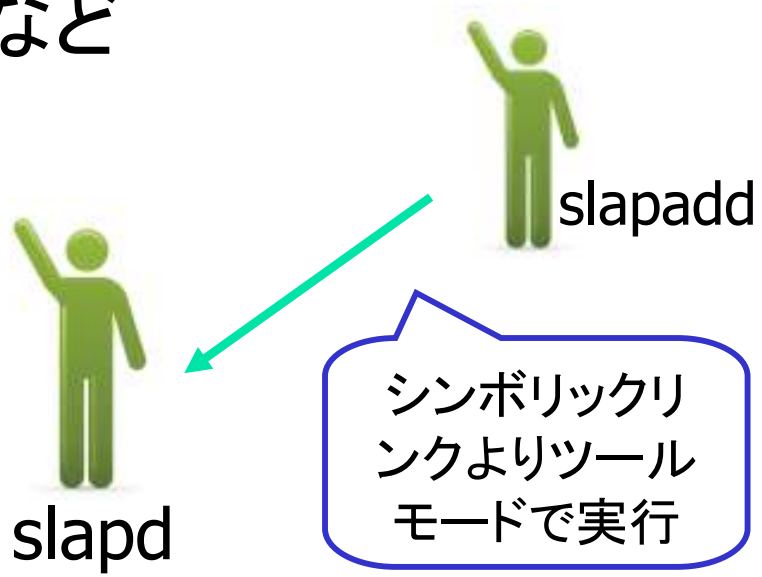
- スレッドライブラリに、目標とする並行度数を指定
  - pthread_setconcurrency() を利用
    - システムリソース消費の抑制などを目的として、アプリケーション側からスレッドの並行度数を絞ることが可能
  - デフォルトは設定なし
    - スレッドライブラリへの並行度数の制限なし

参考

tool-threadsディレクティブ

- slapdを、ツールモードにて実行する場合の最大スレッド数を指定する
  - デフォルト値は、1
  - 設定値は、システムの持つCPUの総数以下に

- slapd -T a や、slapadd など
  - slapadd のほか
  - slapcat
  - slapindex
  - slaptest
  - slapacl…など



ワーカースレッド

で、その先へ！

threadsディレクティブ

- ワーカースレッドの最大数(n)を設定する
  - slapdプロセスは、最大n+1のスレッドを生成

threadsディレクティブ (2)

- デフォルトは、16
  - 2未満では、OpenLDAPサーバの起動不可
  - デフォルト値の2倍(16*2=32)より大きい設定では、起動時に警告メッセージ
    - 33以上が、常に良くないというわけではない
  - 設定可能な最大値は、1024

threadsディレクティブの設定

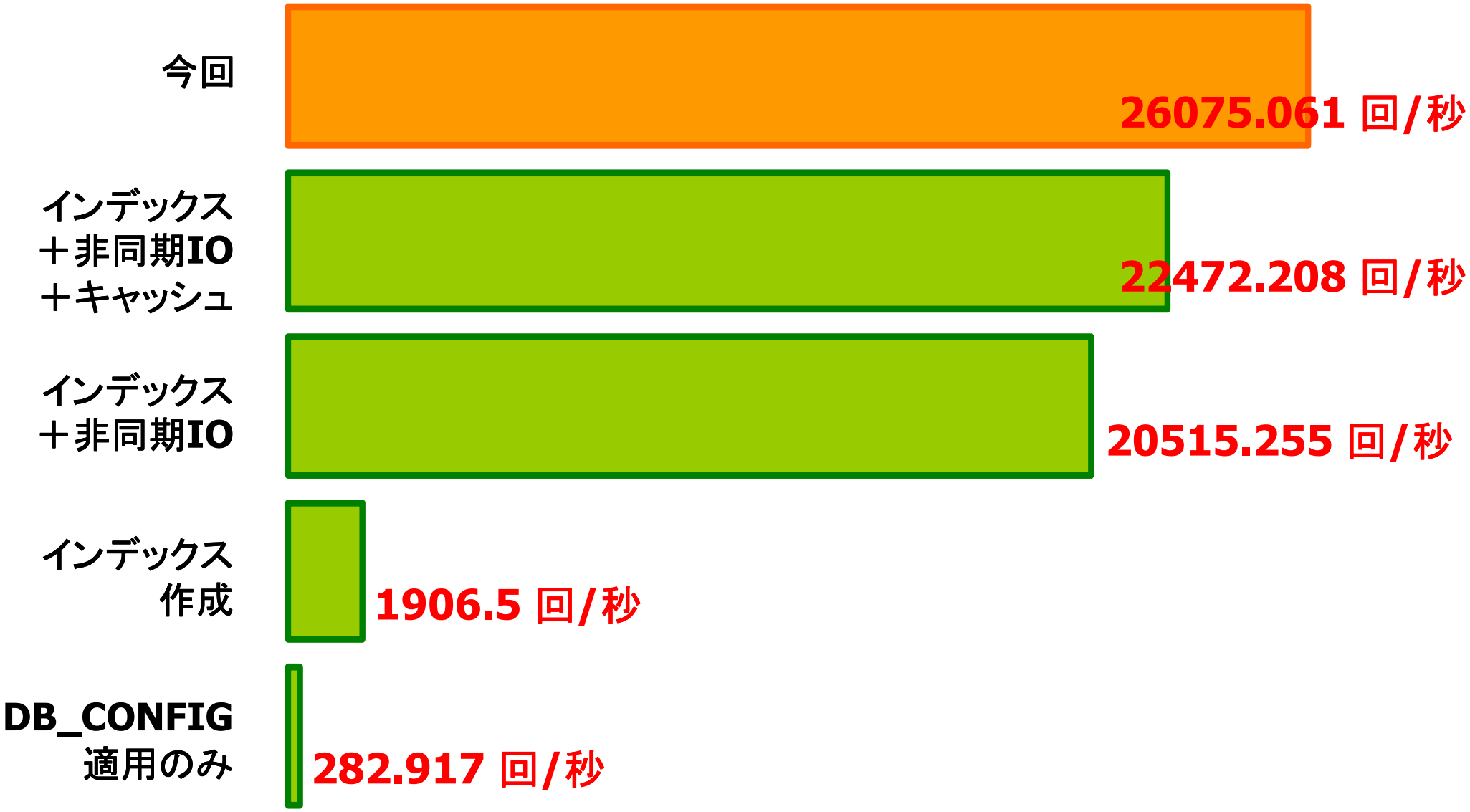
- OpenLDAPコミュニティからの、設定目安
  - 参照系の処理が多い場合
    - ハードウェアのCPUコア数×4、または、×8
    - 最大でも、16程度
  - 更新系の処理が多い場合
    - 16よりも、もっと多く

threadsディレクティブの設定(2)

- 処理内容や環境によって、適正値は異なる
- デフォルト値のスタートも間違いではない
  - まずは、コミュニティの情報を信じても良い

- 可能であれば、実機での確認を！

スレッド数を調整した効果



※ 結果は、今回の検証環境での値です。性能改善のインパクトは、参照系処理、更新系処理の割合や、CPUの特長にも依存します。

スレッド数で、**16%**

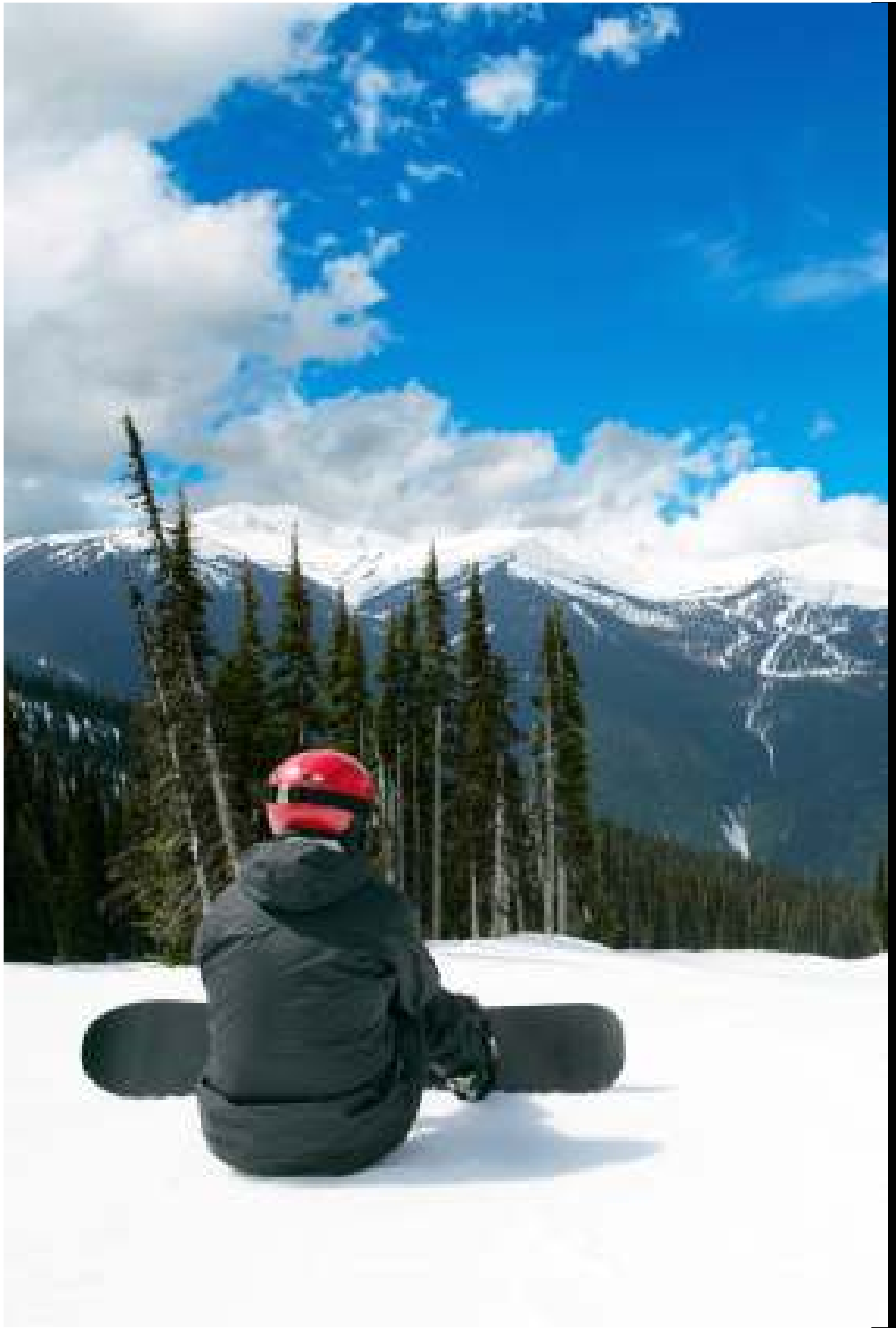
状況に応じ
ワーカースレッド数を調整しよう！

もう一歩、その先へ

様々な要因で、性能は変化

- サイジングを！

1. システムのサービス要件の把握
2. ハード、ミドルウェア特性を考慮したサイジング
3. できる限り本番環境に近い環境での確認



ご清聴、
ありがとうございました